



NXLog User Guide

NXLog Ltd.

Version 4.0.3550, April 2018

Table of Contents

Introduction	1
1. About This Guide.....	2
2. About NXLog	3
2.1. NXLog Features.....	3
2.2. Enterprise Edition.....	5
2.3. What NXLog is Not	7
3. System Architecture	9
3.1. Event Records and Fields.....	9
3.2. Modules and Routes.....	11
3.3. Log Processing Modes	15
4. Available Modules.....	18
4.1. Extension Modules.....	18
4.2. Input Modules.....	19
4.3. Processor Modules.....	21
4.4. Output Modules	21
4.5. Modules by Platform	23
4.6. Modules by Package.....	37
Deployment	65
5. Supported Platforms	66
6. System Requirements	68
7. Red Hat Enterprise Linux & CentOS.....	69
7.1. Installing.....	69
7.2. Upgrading	70
7.3. Uninstalling	70
8. Debian & Ubuntu	71
8.1. Installing.....	71
8.2. Upgrading	72
8.3. Uninstalling	72
9. SUSE Linux Enterprise Server	73
9.1. Installing.....	73
9.2. Upgrading	74
9.3. Uninstalling	74
10. FreeBSD	75
10.1. Installing.....	75
10.2. Upgrading	75
10.3. Uninstalling	76
11. OpenBSD	77
11.1. Installing.....	77
11.2. Upgrading	78
11.3. Uninstalling	78
12. Microsoft Windows.....	79
12.1. Installing.....	79
12.2. Upgrading	82
12.3. Uninstalling	82
12.4. Configure using a custom MSI	83
13. Oracle Solaris	84
13.1. Installing.....	84
13.2. Upgrading	85
13.3. Uninstalling	85
14. IBM AIX.....	86
14.1. Installing.....	86
14.2. Upgrading	86

14.3. Uninstalling	86
15. Apple macOS	87
15.1. Installing	87
15.2. Upgrading	88
15.3. Uninstalling	88
16. Hardening NXLog	89
16.1. Running Under a Non-Root User on Linux	89
16.2. Configuring SELinux	90
16.3. Running Under a Custom Account on Windows	94
17. Relocating NXLog	99
17.1. System V Init File	99
17.2. Systemd Unit File	99
17.3. Edit Configuration File	100
17.4. Modify rpath	100
Configuration	102
18. Configuration Overview	103
18.1. Global Directives	103
18.2. Modules	104
18.3. Routes	104
18.4. Constant and Macro Definitions	105
18.5. Environment Variables	106
18.6. File Inclusion	107
19. NXLog Language	109
19.1. Types	110
19.2. Expressions	111
19.3. Statements	116
19.4. Variables	120
19.5. Statistical Counters	121
20. Reading and Receiving Logs	123
20.1. Receiving over the Network	123
20.2. Reading from a Database	124
20.3. Reading from Files and Sockets	125
20.4. Receiving from an Executable	126
21. Processing Logs	127
21.1. Parsing Various Formats	127
21.2. Alerting	134
21.3. Buffering	136
21.4. Character Set Conversion	136
21.5. Detecting a Dead Agent or Log Source	137
21.6. Event Correlation	138
21.7. Explicit Drop	140
21.8. Extracting Data	141
21.9. Filtering Messages	148
21.10. Format Conversion	148
21.11. Log Rotation and Retention	149
21.12. Message Classification	159
21.13. Parsing Multi-Line Messages	163
21.14. Rate Limiting	165
21.15. Rewriting and Modifying Messages	166
21.16. Timestamps	169
22. Forwarding and Storing Logs	173
22.1. Generating Various Formats	173
22.2. Forwarding over the Network	174
22.3. Sending to Files and Sockets	176

22.4. Storing in Databases	176
22.5. Sending to Executables	177
23. Centralized Log Collection.....	179
23.1. Architecture.....	179
23.2. Collection Modes	180
23.3. Requirements	181
23.4. Data Formats.....	182
24. Encrypted Transfer.....	183
24.1. SSL/TLS Encryption in NXLog	183
24.2. OpenSSL Certificate Creation	185
25. Reliable Message Delivery.....	186
25.1. Crash-Safe Operation	186
25.2. Reliable Network Delivery.....	187
25.3. Protection Against Duplication.....	188
OS Support	191
26. IBM AIX.....	192
27. FreeBSD and OpenBSD	194
28. GNU/Linux.....	197
29. Apple macOS.....	198
30. Oracle Solaris	202
31. Microsoft Windows.....	203
Integration	204
32. Apache HTTP Server.....	205
32.1. Error Log	205
32.2. Access Log	206
33. Apache Tomcat	208
34. APC Automatic Transfer Switch	209
34.1. Configuring via the Web Interface	210
34.2. Configuring via the Command Line	211
35. ArcSight Common Event Format (CEF)	213
35.1. Collecting and Parsing CEF	213
35.2. Generating and Forwarding CEF	214
35.3. Using xm_csv and xm_kvp.....	215
36. AWS CloudWatch	217
36.1. Pulling via API	217
36.2. Pushing Log Data With Lambda.....	220
37. Azure Operations Management Suite	223
37.1. Forwarding Data to Log Analytics	223
37.2. Downloading Data From Log Analytics	226
38. Bro Network Security Monitor	229
38.1. About Bro Logs	229
38.2. Parsing Bro Logs.....	230
39. Brocade Switches	232
40. Checkpoint.....	234
41. Cisco ACS	235
42. Cisco ASA	236
42.1. Forwarding Cisco ASA Logs Over TCP	236
42.2. NetFlow From Cisco ASA	243
43. Common Event Expression (CEE).....	248
43.1. Collecting and Parsing CEE	248
43.2. Generating and Forwarding CEE	249
44. Dell EqualLogic	251
44.1. Configuring via the Group Manager	253
44.2. Configuring via the Command Line	254

45. Dell iDRAC	255
45.1. Configuring via the Web Interface	257
45.2. Configuring via the Command Line	258
46. Dell PowerVault MD Series	260
47. DNS Monitoring	264
47.1. Bind9 on Linux	264
47.2. Windows Server DNS	265
48. Elasticsearch and Kibana.....	271
48.1. Sending Logs to Elasticsearch.....	271
48.2. Forwarding Logs to Logstash	274
49. F5 BIG-IP.....	276
49.1. Collecting BIG-IP Logs via TCP.....	276
49.2. Collecting BIG-IP Logs via UDP	280
49.3. Using SNMP Traps	283
49.4. BIG-IP High Speed Logging	288
50. File Integrity Monitoring	293
50.1. Monitoring on Linux.....	293
Monitoring on Windows	294
51. Graylog.....	297
51.1. Configuring GELF UDP Collection.....	297
51.2. Configuring GELF TCP or TCP/TLS Collection	298
51.3. Collector Sidecar Configuration	301
52. HP ProCurve	304
53. Linux Audit System.....	306
53.1. Audit Rules.....	306
53.2. Using im_linuxaudit	308
53.3. Using auditd Userspace.....	309
54. Linux System Logs	312
54.1. Replacing Rsyslog.....	312
54.2. Forwarding Messages via Socket	313
54.3. Reading Rsyslog Log Files	314
55. Log Event Extended Format (LEEF)	317
55.1. Collecting LEEF Logs.....	317
55.2. Generating LEEF Logs	318
56. Microsoft Exchange	320
56.1. Transport Logs	320
56.2. EventLog	326
56.3. IIS Logs.....	327
56.4. Audit Logs (nxlog-xchg)	327
57. Microsoft IIS.....	334
57.1. Configuring Logging.....	334
57.2. W3C Extended Log File Format.....	335
57.3. IIS Log File Format	337
57.4. NCSA Common Log File Format.....	338
57.5. SMTP Server	339
57.6. Automatic Retrieval of IIS Site Log Locations	341
58. Microsoft SharePoint	343
58.1. Diagnostic Logs.....	343
58.2. Usage and Health Data Logs.....	347
58.3. Audit Logs	352
58.4. Windows EventLog	358
58.5. IIS Logs	358
59. Microsoft SQL Server	359
59.1. Error Log	359

59.2. Audit Log	359
60. Microsoft System Center Operations Manager	365
60.1. Log Types	365
60.2. Collecting Logs	365
61. MongoDB	369
62. Nessus Vulnerability Scanner	371
63. NetApp	376
63.1. Sending Logs in Syslog Format	376
63.2. Sending Logs to a Remote File Share	379
64. .NET Application Logs	381
65. Nginx	385
65.1. Error Log	385
65.2. Access Log	387
66. Postfix	390
66.1. Configuring Postfix Logging	390
66.2. Collecting and Processing Postfix Logs	390
67. Promise	393
67.1. Configuring via Web Interface	394
67.2. Configuring via Command Line	395
68. SafeNet KeySecure	396
68.1. Configuring via the Web Interface	398
68.2. Configuring via the Command Line	399
69. Snare	401
69.1. Collecting Snare	401
69.2. Generating Snare	403
70. Snort	405
71. Splunk	407
71.1. Configuring TCP or UDP Collection	407
71.2. Configuring TLS Collection	409
71.3. Configuring HTTP Event Collection (HEC)	410
72. Synology DiskStation	413
73. Syslog	415
73.1. BSD Syslog (RFC 3164)	415
73.2. IETF Syslog (RFCs 5424-5426)	417
73.3. Collecting and Parsing Syslog	418
73.4. Generating Syslog	422
73.5. Extending Syslog	424
74. Sysmon	428
74.1. Sysmon Events	428
74.2. Collecting Sysmon Events	428
74.3. Advanced Filtering Options	432
75. Ubiquiti UniFi	434
76. VMware vCenter	437
76.1. Local vCenter Logging	437
76.2. Remote vCenter Logging	440
77. Windows EventLog	445
77.1. Local Collection With im_msvisalog	445
77.2. Remote Collection With im_msvisalog	446
77.3. Local Collection With im_mseventlog	447
77.4. Remote Collection With im_wseventing	447
77.5. Remote Collection With im_wmi	448
77.6. Forwarding EventLog	449
78. Windows PowerShell	454
78.1. Using PowerShell Scripts	454

78.2. Logging PowerShell Activity	459
Troubleshooting.....	467
79. NXLog's Internal Logs.....	468
79.1. Log Level	468
79.2. Injecting Internal Logs into a Route.....	468
79.3. Running NXLog in the Foreground	468
79.4. Generating Additional Log Messages in your Configuration	468
80. Resolving Common Issues	470
80.1. "Nxlog failed to start".....	470
80.2. "Permission denied".....	470
80.3. Log Entries are Concatenated with Logstash.....	470
80.4. Log File is in Use by Another Application	470
80.5. "Connection refused" with im_tcp or im_ssl.....	470
80.6. "Missing logdata"	471
80.7. Windows EventLog Error: "ignoring source"	472
80.8. Processing Stops if an Output Fails	472
81. Debugging Data Processing	473
82. Debugging NXLog.....	476
82.1. Memory Leaks.....	476
Enterprise Edition Reference Manual.....	477
83. Man Pages.....	478
83.1. nxlog(8)	478
83.2. nxlog-processor(8)	480
84. Configuration	482
84.1. General Directives	482
84.2. Global Directives.....	483
84.3. Common Module Directives.....	486
84.4. Route Directives	491
85. Language	494
85.1. Types	494
85.2. Expressions	494
85.3. Statements	503
85.4. Variables	505
85.5. Statistical Counters.....	505
85.6. Functions	506
85.7. Procedures	511
86. Extension Modules.....	514
86.1. Remote Management (xm_admin)	514
86.2. AIX Auditing (xm_aixaudit).....	520
86.3. Apple System Logs (xm_asl)	521
86.4. Basic Security Module Auditing (xm_bsm)	523
86.5. CEF (xm_cef)	528
86.6. Character Set Conversion (xm_charconv).....	530
86.7. CSV (xm_csv)	532
86.8. External Program Execution (xm_exec).....	536
86.9. Filelist (xm_filelist).....	538
86.10. File Operations (xm_fileop).....	539
86.11. GELF (xm_gelf).....	542
86.12. Grok (xm_grok).....	546
86.13. JSON (xm_json)	548
86.14. Key-Value Pairs (xm_kvp).....	552
86.15. LEEF (xm_leef)	561
86.16. Microsoft DNS Server (xm_msdns)	562
86.17. Multi-Line Message Parser (xm_multiline)	565

86.18. NetFlow (xm_netflow)	573
86.19. NPS (xm_nps)	574
86.20. Pattern Matcher (xm_pattern)	576
86.21. Perl (xm_perl)	580
86.22. Python (xm_python)	583
86.23. Resolver (xm_resolver)	586
86.24. Rewrite (xm_rewrite)	588
86.25. Ruby (xm_ruby)	591
86.26. SNMP Traps (xm_snmp)	593
86.27. Remote Management (xm_soapadmin)	597
86.28. Syslog (xm_syslog)	601
86.29. W3C (xm_w3c)	613
86.30. WTMP (xm_wtmp)	617
86.31. XML (xm_xml)	618
87. Input Modules	622
87.1. Fields	622
87.2. BSD/Linux Process Accounting (im_acct)	622
87.3. AIX Auditing (im_aixaudit)	624
87.4. Azure (im_azure)	625
87.5. Batched Compression over TCP or SSL (im_batchcompress)	627
87.6. Basic Security Module Auditing (im_bsm)	629
87.7. CheckPoint OPSEC LEA (im_checkpoint)	630
87.8. DBI (im_dbi)	634
87.9. Event Tracing for Windows (im_etw)	636
87.10. Program (im_exec)	639
87.11. File (im_file)	640
87.12. File Integrity Monitoring (im_fim)	643
87.13. HTTP(s) (im_http)	646
87.14. Internal (im_internal)	648
87.15. Kafka (im_kafka)	650
87.16. Kernel (im_kernel)	651
87.17. Linux Audit System (im_linuxaudit)	652
87.18. Mark (im_mark)	654
87.19. MS EventLog for Windows XP/2000/2003 (im_mseventlog)	655
87.20. MS EventLog for Windows 2008/Vista and Later (im_msvisalog)	658
87.21. Null (im_null)	664
87.22. OCI (im_oci)	664
87.23. ODBC (im_odbc)	665
87.24. Perl (im_perl)	667
87.25. Python (im_python)	669
87.26. Redis (im_redis)	671
87.27. Windows Registry Monitoring (im_regmon)	672
87.28. Ruby (im_ruby)	675
87.29. TLS/SSL (im_ssl)	677
87.30. TCP (im_tcp)	679
87.31. UDP (im_udp)	680
87.32. Unix Domain Socket (im_uds)	682
87.33. Windows Performance Counters (im_winperfcount)	683
87.34. Windows Management Instrumentation (im_wmi)	685
87.35. Windows Event Collector (im_wseventing)	687
87.36. ZeroMQ (im_zmq)	695
88. Processor Modules	697
88.1. Blocker (pm_blocker)	697
88.2. Buffer (pm_buffer)	698

88.3. Event Correlator (pm_evcrr)	700
88.4. Filter (pm_filter)	706
88.5. HMAC Message Integrity (pm_hmac)	706
88.6. HMAC Message Integrity Checker (pm_hmac_check)	708
88.7. Message De-Duplicator (pm_norepeat)	710
88.8. Null (pm_null)	711
88.9. Pattern Matcher (pm_pattern)	712
88.10. Message Format Converter (pm_transformer)	715
88.11. Timestamping (pm_ts)	717
89. Output Modules	721
89.1. Batched Compression over TCP or SSL (om_batchcompress)	721
89.2. Blocker (om_blocker)	723
89.3. DBI (om_db)	724
89.4. Elasticsearch (om_elasticsearch)	726
89.5. EventDB (om_eventdb)	729
89.6. Program (om_exec)	731
89.7. File (om_file)	732
89.8. HTTP(s) (om_http)	734
89.9. Kafka (om_kafka)	736
89.10. Null (om_null)	738
89.11. Oracle (om_oci)	738
89.12. ODBC (om_odbc)	739
89.13. Perl (om_perl)	741
89.14. Python (om_python)	743
89.15. Redis (om_redis)	744
89.16. Ruby (om_ruby)	745
89.17. TLS/SSL (om_ssl)	747
89.18. TCP (om_tcp)	749
89.19. UDP (om_udp)	751
89.20. UDP with IP Spoofing (om_udpspoof)	752
89.21. UDS (om_uds)	754
89.22. WebHDFS (om_webhdfs)	755
89.23. ZeroMQ (om_zmq)	757
NXLog Manager	759
90. Introduction	760
90.1. Requirements	760
90.2. Architecture	760
91. Installation	761
91.1. Installing on Debian Wheezy	761
91.2. Installing on Redhat 6 & 7	761
91.3. Installing as Docker application	761
91.4. Configuring NXLog Manager for standalone mode	762
91.5. Configuring NXLog Manager for cluster mode	762
91.6. Database initialization	766
91.7. NXLog Agent installation	767
91.8. NXLog Manager configuration	768
91.9. Enabling HTTPS for NXLog Manager	775
91.10. Upgrading NXLog Manager	778
91.11. Host Setup Common Issues	778
92. Dashboard and the Menu	781
92.1. Logging in	781
92.2. The menu bar	781
92.3. Dashboard	783
93. Fields	785

94. Patterns	788
94.1. Pattern Groups	788
94.2. Creating a pattern	790
94.3. Message classification with patterns.....	792
94.4. Searching patterns	793
94.5. Exporting and importing patterns	794
94.6. Using patterns.....	794
95. Correlation.....	795
95.1. Correlation rulesets	795
95.2. Correlation rules.....	795
95.3. Exporting and importing correlation rules	797
95.4. Using correlation rules	798
96. Agents	799
96.1. Agent information	804
96.2. Agent configuration	806
97. Templates	812
97.1. Template configuration	813
98. Agent groups.....	816
98.1. Agent list in a group	816
99. Certificates.....	818
99.1. Listing certificates.....	818
99.2. Creating a CA.....	819
99.3. Creating a certificate	819
99.4. Exporting	820
99.5. Importing.....	821
99.6. Revoking and deleting certificates	821
99.7. Renewing certificates.....	821
100. Settings	822
100.1. Agent Manager	822
100.2. Certificates.....	824
100.3. Mail.....	824
100.4. Config backup	825
100.5. License.....	825
100.6. User Settings.....	826
101. Users, roles and access control	828
101.1. Users	828
101.2. Roles.....	830
101.3. Audit Trail	831
102. RESTful Web Services.....	833
102.1. Agentmanager	833
102.2. Appinfo	833
102.3. Agentinfo.....	834
102.4. Addagent	834
102.5. Modifyagent	835
102.6. Deleteagent.....	835
102.7. Certificateinfo	836
102.8. Createfield	836

Introduction

Chapter 1. About This Guide

This guide is designed to give you all the information and skills you need to successfully deploy and configure NXLog in your organization.

NXLog is available in two versions, the *Community Edition* and the *Enterprise Edition*. Features that are unique to the *Enterprise Edition* are noted as such, except in the [Reference Manual](#) (the *Community Edition* Reference Manual is published separately). See the [NXLog Features](#) and [Enterprise Edition](#) sections for more details regarding the functionality provided by the two NXLog editions.

WARNING

Though most of the content applies to all versions of NXLog Community Edition and NXLog Enterprise Edition, this guide was written specifically for **NXLog Enterprise Edition** version **4.0.3550**. Some features covered by this document may not be available in earlier versions of NXLog and earlier versions of NXLog may behave differently than documented here.

This guide is divided into the following parts.

- [Introduction](#) gives an overview of NXLog and its architecture.
- [Deployment](#) provides instructions for installing NXLog and setting up options for additional security.
- [Configuration](#) introduces the extensive configuration system and the NXLog language.
- [OS Support](#) lists the ways that NXLog can be configured to integrate with specific operating systems.
- [Integration](#) provides detailed information about integrating NXLog with various applications, devices, and protocols.
- [Troubleshooting](#) provides suggestions for troubleshooting, and gives explanations for common problems you may encounter.
- The [Enterprise Edition Reference Manual](#) contains a complete, detailed listing of all available configuration directives, functions, procedures, and NXLog language constructs in NXLog Enterprise Edition.
- [NXLog Manager](#) documents the installation and use of the NXLog Manager.

WARNING

If you would like to copy/paste configuration content from the Guide, please do so using the HTML format. It is not possible to guarantee appropriate selection behavior with the PDF format.

Chapter 2. About NXLog

Modern IT infrastructure produces large volumes of event logging data. In a single organization, hundreds or thousands of different devices, applications, and appliances generate event log messages. These messages require many log processing tasks, including filtration, classification, correlation, forwarding, and storage. In most organizations these requirements are met with a collection of scripts and programs, each with its custom format and configuration. NXLog provides a single, high-performance, multi-platform product for solving all of these tasks and achieving consistent results.

At NXLog's inception, there were various logging solutions available, but none with the required features. Most were single threaded and Syslog-oriented, without native support for Windows. Work on NXLog began with the goal of building a modern logger with a multi-threaded design, a clear configuration syntax, multi-platform support, and clean source code. NXLog was born in 2009 as a closed source product heavily used in several production deployments. The source code of NXLog Community Edition was released in November 2011.

NXLog can process event logs from thousands of different sources with volumes over 100,000 events per second. It can accept event logs over TCP, TLS/SSL, and UDP; from files and databases; and in Syslog, Windows EventLog, and JSON formats. NXLog can also perform advanced processing on log messages, such as rewriting, correlating, alerting, pattern matching, scheduling, and log file rotation. It supports prioritized processing of certain log messages, and can buffer messages on disk or in memory to work around problems with input latency or network congestion. After processing, NXLog can store or forward event logs in any of many supported formats. Inputs, outputs, log formats, and complex processing are implemented with a modular architecture and a powerful configuration language.

2.1. NXLog Features

This section gives an overview of some of the key advantages of NXLog over alternative systems.

Multi-platform deployment

Installer packages are provided for multiple platforms, including Linux, Windows, and Android. You can use NXLog across your infrastructure, without resorting to different tools for different platforms.

Client or server operation

Depending on the configuration, NXLog will run as a server, a client, or a combination of both. You have the freedom to choose the deployment architecture that best meets your needs. For example, NXLog can collect local log data and forward it, relay data without storing it locally, or save incoming log data to disk.

Many input and output types and formats

NXLog can accept data from many different sources, convert the data internally, and output it to other destinations. You can use NXLog as a single tool to process all of the different types of logs in your organization. For example, logs can be collected from files, databases, Unix domain sockets, network connections, and other sources. BSD Syslog, IETF Syslog, the Snare Agent format, Windows EventLog, JSON, and other formats are supported. NXLog can likely be configured to read or write logs in your custom application format, using the NXLog language and provided extension modules.

High performance, scalable architecture

With an event-based architecture for processing tasks in parallel, non-blocking input and output where possible, and a worker thread pool for incoming log messages, NXLog is designed for high performance on modern multi-core and multi-processor systems. The input/output readiness notifications provided by most operating systems are used to efficiently handle large numbers of open files and network connections.

Security

NXLog provides features throughout the application to maintain the security of your log data and systems. The core can be configured to run as an unprivileged user, and special privileges (such as binding to ports below 1024) are accessed through Linux capabilities rather than requiring the application to run as root. TLS/SSL is supported for encrypted, authenticated communications and to prevent data interception or alteration during transmission.

Modular architecture

NXLog has a lightweight modular architecture, providing a reduced memory footprint and increased flexibility for different uses. The core handles files, events, and sockets, and provides the configuration language; modules provide the input, output, and processing capabilities. Because modules use a common API, you can write new modules to extend the features of NXLog.

Message buffering

Log messages can be buffered in memory or on disk. This increases reliability by holding messages in a temporary cache when a network connectivity issue or dropout occurs. Conditional buffering can be configured by using the NXLog language to define relevant conditions. For example, UDP messages may arrive faster than they can be processed, and NXLog can buffer the messages to disk for processing when the system is under less load. Conditional buffering can be used to explicitly buffer log messages during certain hours of the day or when the system load is high.

Prioritized processing

NXLog can be configured to separate high-priority log processing from low-priority log processing, ensuring that it processes the most important data first. When the system is experiencing high load, NXLog will avoid dropping important incoming messages. For example, incoming UDP messages can be prioritized to prevent dropped logs if a high volume of TCP messages overloads the system.

Message durability

Built-in flow control ensures that a blocked output does not cause dropped log messages when buffers are full. In combination with the previously mentioned parallel processing, buffering, and prioritization, the possibility of message loss is greatly reduced.

Familiar and powerful configuration syntax

NXLog uses an Apache-style configuration syntax that is easy to read and can be parsed or generated with scripts. The NXLog language supports advanced scripting and processing capabilities that are usually only found in full-fledged scripting languages. The syntax is similar to Perl, so users familiar with that language can learn it easily. It supports polymorphic functions and procedures and regular expressions with captured substrings. Modules can register additional functions and procedures to further extend the capabilities of the language.

Scheduled tasks and log rotation

NXLog includes a scheduler service. A task can be scheduled from any module without requiring an external tool such as Cron. Log files can be rotated automatically, on a time-based schedule or according to file size. The file reader and writer modules in NXLog can detect when an input or output file moves or changes its name, and re-open the file automatically.

Advanced message processing

NXLog can perform advanced processing actions on log messages, in addition to the core features already mentioned. By using additional modules, NXLog can solve many related tasks such as message classification, event correlation, pattern matching, message filtering, message rewriting, and conditional alerting. You can use a single tool for all log processing functionality.

Offline processing

Sometimes log messages need to be processed in batches for conversion, filtering, or analysis. NXLog provides an offline mode in which it processes all input and then exits. Because NXLog does not assume that the event time and processing time are identical, time-based correlation features can be used even during offline log processing.

International character and encoding support

NXLog supports explicit character set conversion and automatic character set detection. Log messages received in different character sets can be automatically normalized to a common standard, allowing messages to be compared across different sources.

2.2. Enterprise Edition

While the NXLog Community Edition provides all the flexibility and performance of the NXLog engine, the NXLog Enterprise Edition provides additional enhancements, including modules and core features, as well as regular hot-fixes and updates. The Enterprise Edition provides the following enhancements.

Additional platform support

In addition to Linux, Windows, and Android, installer packages are provided for the BSDs and the major variants of Unix (AIX, Solaris, and macOS).

Signed installer packages

Installer packages are certificate signed to ensure that the binaries are not corrupted or compromised.

On-the-wire compression

Log data can be transferred in compressed batches with the [im_batchcompress](#) and [om_batchcompress](#) input/output modules. This can help in limited bandwidth scenarios.

UDP source IP address spoofing

Some SIEM and log collection systems use the IP address of the UDP Syslog packet sent by the client. When used as a server or relay, the [om_udpspoof](#) output module can be configured to retain the original IP address of the sender.

Better control over SSL and TLS

Due to vulnerabilities discovered in the SSL protocols, some protocols may need to be disabled. The various SSL/TLS networking modules in NXLog Enterprise Edition can be configured to allow only specific protocols via the `SSLProtocol` directive. On Windows, NXLog Enterprise Edition can utilize TLSv1.2 while NXLog Community Edition supports TLSv1.0 only.

ODBC input and output

The ODBC input and output modules, [im_odbc](#) and [om_odbc](#), allow log data to be read from or inserted into any ODBC compliant database. The primary purpose of the `im_odbc` module is native Windows MSSQL support to enable log collection from Windows applications that write logs to MSSQL. The `om_odbc` output module can be used to insert data into an ODBC database. These modules are available on Windows as well as Linux.

Support for trusted timestamping

The Time-Stamp Protocol, defined by RFC 3161, uses a Time-Stamp Authority to cryptographically sign each message. This signature can be used later to prove that the message existed at that time and has not been modified since. The [pm_ts](#) processor module provides support for the Time-Stamp Protocol.

Message integrity protection

The [pm_hmac](#) and [pm_hmac_check](#) processor modules provide chained HMAC-based integrity protection for tamper-proof logs. The `pm_hmac` module signs outgoing messages while the `pm_hmac_check` module verifies incoming messages.

Remote management

A dedicated [xm_soapadmin](#) extension module allows NXLog agents to be managed remotely over a secure SOAP/SSL connection or integrated with existing monitoring and management tools. The configuration, correlation rules, patterns, and certificates can all be updated remotely from the NXLog Manager web interface or from scripts. In addition, the NXLog agent and the individual modules can be stopped/started and log collection statistics can be queried for real-time statistics.

Crash recovery

Additional functionality is provided to guarantee a clean recovery in the case of a system crash, ensuring that no messages are lost or duplicated.

Event correlation

The [pm_evcrr](#) processor module can efficiently solve complex event correlation tasks, with capabilities similar to what the open-source SEC tool provides.

HTTP(S) protocol support

RESTful services are becoming increasingly popular, even for logging. The [im_http](#) and [om_http](#) input and output modules make it possible to send or receive log message data over HTTP or HTTPS.

File integrity and registry monitoring

Several compliance standards mandate file integrity monitoring. With the [im_fim](#) input module, NXLog Enterprise Edition can be used to detect modifications to files or directories. This module is available on Windows as well as Linux. The [im_regmon](#) module provides monitoring of the Windows Registry.

Structured data formats

The [xm_xml](#) extension module can parse nested XML and data stored in XML attributes. Parsing of nested JSON has also been implemented in [xm_json](#), and UTF-8 validation can be enforced to avoid parser failures caused by invalid UTF-8 from other tools.

Native W3C parser

The W3C format is widely used in various Microsoft products and perhaps IIS is the most well-known producer. Parsing of W3C is possible with the [xm_csv](#) extension module, but that requires defining the fields in the configuration and adjustment when the IIS configuration is changed. The [xm_w3c](#) extension module can automatically parse the logs using the field information stored in the headers. It also supports automatic parsing of the data format produced by BRO.

More support for SIEM products

The [xm_cef](#) and [xm_leef](#) modules provide parsing and generation of CEF and LEEF formatted data. CEF (Common Event Format) was introduced by HP ArcSight and LEEF (Log Event Extended Format) is used by IBM Security QRadar.

Simplified data processing configuration

Two extension modules help simplify the configuration. The [xm_rewrite](#) module allows fields to be renamed, kept (whitelisted), or deleted (blacklisted). It also supports the Exec directive so log processing logic can be localized and to avoid duplicated statements. The [xm_filelist](#) module provides two functions, [contains\(\)](#) and [matches\(\)](#), which can be invoked to check whether a string is present in a text file. This can be a username, IP address, or similar. The files are cached in memory and any changes are automatically picked up without the need to reload NXLog.

Custom input and output modules in Perl

Perl has a vast number of libraries that can be used to easily implement integration with various APIs, formats, and protocols. The [im_perl](#) and [om_perl](#) input and output modules make it possible to utilize Perl to collect and output data without the need to run the code as an external script.

Name resolution

The [xm_resolver](#) extension module provides cached DNS lookup functions for translating between IP addresses and host names. User and group names can also be mapped to/from user and group ids.

Elasticsearch integration

The [om_elasticsearch](#) output module allows log data to be loaded directly into an Elasticsearch server without requiring Logstash.

Checkpoint LEA input

The [im_checkpoint](#) input module enables the remote collection of Checkpoint firewall logs over the OPSEC/LEA protocol. This feature is only available in the Linux version.

Remote Windows EventLog collection

The [im_wmi](#) module allows remote collection of Windows EventLog over the WMI protocol on Linux hosts

without the need to install an agent on the Windows target. This module is only available in the Linux version. In addition, the [im_msvisalog](#) module can query and collect Windows EventLog remotely over MSRPC on Windows Vista and later versions.

Redis Support

Redis is often used as an intermediate queue for log data. Two native modules, [im_redis](#) and [om_redis](#), are available to push data to and pull data from Redis servers.

SNMP input

The [xm_snmp](#) extension module can be used to parse SNMP traps. The traps can then be handled like regular log messages: converted to Syslog, stored, forwarded, etc.

Multi-platform support for Windows Event Forwarding

The [im_wseventing](#) input module can be used to collect forwarded events from Windows hosts. The Windows clients can be configured from Group Policy to send Windows EventLog using Windows Event Forwarding. While NXLog Enterprise Edition can collect Windows EventLog remotely over WMI and MSRPC, this module provides improved security for collecting from Windows machines in agent-less mode, with support for both Kerberos and HTTPS data transfer. The *im_wseventing* module is platform independent and available on Linux as well as Windows.

HDFS output

The [om_webhdfs](#) output module is available to support the Hadoop ecosystem.

Windows Performance Counters

The [im_winperfcount](#) input module can collect metrics from Windows Performance Counters such as CPU, disk, and memory statistics.

Reading Windows EventLog files directly

The [im_msvisalog](#) module can read `.evt`, `.evtx`, and `.etl` EventLog files directly; this is particularly useful for forensics purposes.

Additional Windows EventLog data

The [im_msvisalog](#) module retrieves the *EventData* and *UserData* parts which can contain important data in some log sources. In addition, SID values in the EventLog *Message* can be resolved to account names to produce the same output that EventViewer gives.

Netflow support

The [xm_netflow](#) extension module can parse Netflow packets received over UDP. It supports Netflow v1, v5, v7, v9, and IPFIX.

ZeroMQ support

ZeroMQ is a popular high performance messaging library. The [im_zmq](#) and [om_zmq](#) modules provide input and output support for the ZeroMQ protocol.

Regular hot fixes

Unlike NXLog Community Edition which is a volunteer effort, NXLog Enterprise Edition receives regular hot fixes and enhancements.

2.3. What NXLog is Not

NXLog provides a broad range of features for collecting, processing, forwarding, and storing log data. However, NXLog is *not* a SIEM product and does not provide:

- a graphical interface (or "dashboard") for searching logs and displaying reports,
- vulnerability detection or integration with external threat data,
- automatic analysis and correlation algorithms, or

- pre-configured compliance and retention policies.

NXLog *does* provide processing features that can be used to set up analysis, correlation, retention, and alerting; NXLog *can* be integrated with many other products to provide a complete solution for aggregation, analysis, and storage of log data.

Chapter 3. System Architecture

NXLog's architecture utilizes loadable modules that provide input, output, and parsing functionality. Different modules can be used together to create log data routes that meet the requirements of the logging environment. Each route accepts log messages in a particular format, processes or transforms them, and then outputs them in one of the supported formats. Routes work in parallel, so processing of different types of log data can happen at the same time.

The NXLog core is responsible for parsing the configuration file, monitoring files and sockets, managing internal events, and coordinating log data queues and modules according to the configured routes. Files and sockets are added to the core by the various modules, and the core delegates events when necessary. Modules also dispatch log events to the core, which passes each one to the appropriate module. In this way, the core can centrally control all events and the order of their execution making prioritized processing possible. Each event belonging to the same module instance is executed in sequential order, not concurrently. This ensures that message order is kept and allows modules to be written without concern for concurrency. Yet because the modules run concurrently, the global log processing flow remains parallelized.

3.1. Event Records and Fields

A log message is an *event*, and the data relating to that event is collectively an *event record*. When NXLog processes an event record, it stores the various values in *fields*. The following sections describe *event records* and *fields* in the context of NXLog processing.

3.1.1. Event Records

There are many kinds of event records. A few important ones are listed here.

- The most common event record is a single line. Thus the default is [LineBased](#) for the [InputType](#) and [OutputType](#) directives.
- It is also common for an event record to use a single UDP datagram. NXLog can send and receive UDP events with the [im_udp](#) and [om_udp](#) modules.
- Some event records are generated using multiple lines. These can be joined into a single event record with the [xm_multiline](#) module.
- Event records may be stored in a database. Each row in the database represents an event. In this case the [im_odbc](#) and [om_odbc](#) modules can be used.
- It is common for structured event records to be formatted in CSV, JSON, or XML formats. The [xm_csv](#), [xm_json](#), and [xm_xml](#) modules provide functions and procedures for parsing these.
- NXLog provides a [Binary InputType](#) and [OutputType](#) for use when compatibility with other logging software is not required. This format preserves parsed fields and their types.

In NXLog, each event record consists of the raw event data (in a field named [\\$raw_event](#)) and additional fields generated during processing and parsing.

3.1.2. Fields

All event log messages contain important data such as user names, IP addresses, and application names. Traditionally, these logs have been generated as free form text messages prepended by basic metadata like the time of the event and a severity value.

While this format is easy for humans to read, it is difficult to perform log analysis and filtering on thousands of free-form logs. In contrast, *structured logging* provides means for matching messages based on key-value pairs. With structured logging, an event is represented as a list of key-value pairs. The name of the field is the key and the field data is the value. NXLog's core design embraces structured logging. Using various features provided by NXLog, a message can be parsed into a list of key-value pairs for processing or as part of the message sent to the destination.

When a message is received by NXLog, it creates an internal representation of the log message using fields. Each field is **typed** and represents a particular attribute of the message. These fields pass through the log route, and are available in each successive module in the chain, until the log message has been sent to its destination.

1. The special **\$raw_event** field contains the raw data received by the input module. Most input and output modules only transfer **\$raw_event** by default.
2. The core adds a few **additional fields** by default. This includes, for example, the **\$EventReceivedTime** field which is set to the time when the event is received.
3. The input module may add other fields. For example, the **im_udp** module adds a **\$MessageSourceAddress** field.
4. Some input modules, such as **im_msvisalog** and **im_odbc**, map fields from the source directly to fields in the NXLog event record.
5. Parsers such as the **parse_syslog()** procedure will add **more fields**.
6. Custom fields can be added by using the NXLog language and an **Exec** directive.
7. The NXLog language or the **pm_pattern** module can be used to set fields using regular expressions. See [Extracting Data](#).

When the configured output module receives the log message, in most cases it will use the contents of the **\$raw_event** field only. If the event's fields have been modified, it is therefore important to update **\$raw_event** from the other fields. This can be done with the NXLog language, perhaps using a procedure like [to_syslog_bsd\(\)](#).

A field is denoted and referenced in the configuration by a preceding dollar sign (**\$**). See the [Fields](#) section in the Reference Manual for more information.

Example 1. Sample Syslog Event as Processed by NXLog

This example shows a Syslog event and its corresponding fields as processed by NXLog.

Syslog Message

```
<38>Nov 21 11:40:27 myhost sshd[8459]: Accepted publickey for john from 192.168.1.22 port 41193
ssh2←
```

The below configuration parses the event with the `parse_syslog()` procedure, which adds a set of `fields` to the event record.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input udp>
6   Module im_udp
7   Host   0.0.0.0
8   Port   514
9   Exec   parse_syslog();
10 </Input>
```

When the above message is parsed by `parse_syslog()`, the following fields are added to the event record.

Table 1. Syslog Fields as Parsed by parse_syslog()

Field	Value
\$Message	Accepted publickey for john from 192.168.1.22 port 41193 ssh2
\$SyslogSeverityValue	6
\$SyslogSeverity	INFO
\$SeverityValue	2
\$Severity	INFO
\$SyslogFacilityValue	4
\$SyslogFacility	AUTH
\$EventTime	2016-11-21 11:40:27
\$Hostname	myhost
\$SourceName	sshd
\$ProcessID	8459

See [Extracting Data](#) for more examples.

3.2. Modules and Routes

NXLog uses loadable *modules* (plugins) that extend its capabilities to accept, process, and output log messages. The log processing path is made of *module instances* connected by *routes*. The following sections explain this in greater detail.

3.2.1. Modules

A *module* is a `foo.so` or `foo.dll` that can be loaded by the NXLog core. The various modules provide different capabilities to NXLog. A module *instance* is part of the configured data flow. For example, the configuration block

for an input module instance begins with `<Input instancename>`. See the *Instance* examples below. A single module can be used in multiple instances. With regard to configuration, a module *instance* is often referred to as simply a *module*.

There are four types of modules.

Input

Functionality for accepting or retrieving log data is provided by input modules. An input module instance is a source or producer. It accepts log data from a source and produces event records.

An Input Module Instance

```
1 <Input foo_in>
2   Module im_foo
3   ...
4 </Input>
```

Output

Output modules provide functionality for sending log data to a local or remote destination. An output module instance is a sink, destination, or consumer. It is responsible for consuming event records produced by one or more input module instances.

An Output Module Instance

```
1 <Output foo_out>
2   Module om_foo
3   ...
4 </Output>
```

Extension

The NXLog language can be extended with extension modules. Extension module instances do not process log data directly. Instead, they provide features (usually functions and procedures) that can be used from other parts of the processing pipeline. Many extension module instances require no directives other than the [Module](#) directive.

Example 2. Using an Extension Module

In this example, the `xm_syslog` module is loaded by the [Extension](#) block. This module provides the `parse_syslog()` procedure, in addition to other functions and procedures. In the following [Input](#) instance, the `Exec` directive calls `parse_syslog()` to parse the Syslog-formatted event.

nxlog.conf

```
1 <Extension xm_syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input in>
6   Module im_file
7   File   '/var/log/messages'
8   Exec   parse_syslog();
9 </Input>
```

Processor

Processor modules offer features for transforming, filtering, or converting log messages. One or more processor module instances can be used in a route between input and output module instances.

A Processor Module Instance

```
1 <Processor foo>
2     Module pm_foo
3     ...
4 </Processor>
```

NOTE

Many processing functions and procedures are available through the NXLog language and can be accessed through the `Exec` directive in an Input or Output block without using a separate processor module instance. However, a separate processor module (`pm_null`, perhaps) will use a separate worker thread, providing additional processing parallelization.

For a list of available modules, see [Available Modules](#).

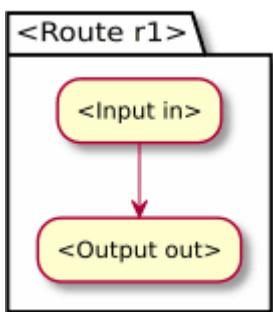
3.2.2. Routes

Most log processing solutions are built around the same concept. The input is read from a source, log messages are processed, and then log data is written to a destination. In NXLog, this path is called a "route" and is configured with a Route block.

Routes are made up of one or more inputs, zero or more processors, and one or more outputs.

Example 3. A Simple Route

This route accepts input with the `in` module and sends it to the `out` module. This is the simplest functional route.

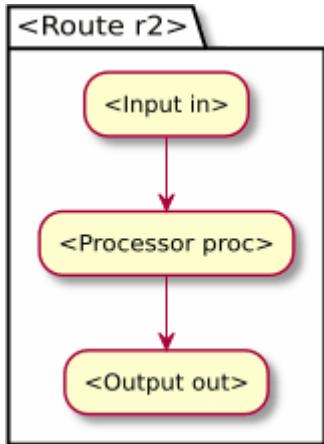


nxlog.conf

```
1 <Route r1>
2     Path      in => out
3 </Route>
```

Example 4. A Route With a Processor

This route extends the previous example by adding an intermediate processing module `proc`.



nxlog.conf

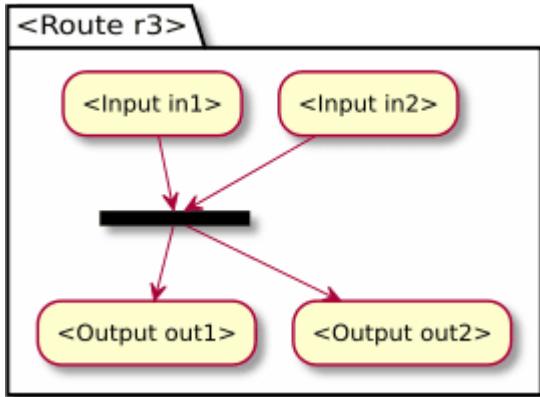
```

1 <Route r2>
2     Path      in => proc => out
3 </Route>

```

Example 5. Advanced Route With Multiple Input/Output Modules

This route uses two input modules and two output modules. Input from `in1` and `in2` will be combined and sent to both `out1` and `out2`.



nxlog.conf

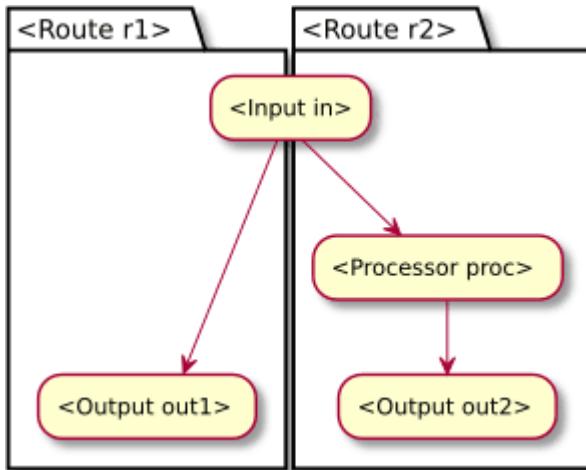
```

1 <Route r3>
2     Path      in1, in2 => out1, out2
3 </Route>

```

Example 6. Branching: Two Routes Using One Input Module

A module can be used by multiple routes simultaneously, as in this example. The `in1` module instance is only declared once, but is used by both routes.



`nxlog.conf`

```

1 <Route r1>
2     Path      in => out1
3 </Route>
4
5 <Route r2>
6     Path      in => proc => out2
7 </Route>

```

3.3. Log Processing Modes

NXLog can process logs in three modes. Each mode has different characteristics, and you can use any combination of modes for your overall logging infrastructure.

- **Agent-Based Collection:** NXLog runs on the system that is generating the log data.
- **Agent-Less Collection:** Hosts or devices generate log data and send it over the network to NXLog.
- **Offline Log Processing:** The [nxlog-processor\(8\)](#) tool performs batch log processing.

3.3.1. Agent-Based Collection

With agent-based collection, NXLog runs as an *agent* on the system that is generating the log data. It collects the log data and sends it to another NXLog instance over the network.

NOTE

We recommend agent-based log collection for most use cases. In particular, we recommend this mode if you need strong security and reliability or need to transform log data before it leaves the system on which it was generated.

Agent-based log collection offers several important advantages over agent-less collection.

- Log data can be collected from more sources. For example, you can collect logs directly from files, instead of relying on a logging process to send log data across the network.
- NXLog's processing features are available. You can filter, normalize, and rewrite log data before sending it to a destination, whether a NXLog instance or a log aggregation system. This includes the ability to send structured log data, such as JSON and key-value pairs.

- You have full control over the transfer of the log data. Messages can be sent using a variety of protocols, including over TLS/SSL encrypted connections for security. Log data can be sent in compressed batches and can be buffered if necessary.
- Log collection in this mode is more reliable. NXLog includes delivery guarantees and flow control systems that make sure your log data gets to its destination. You can monitor the health of the NXLog agent to check that it is operating correctly.

Although agent-based collection has many compelling advantages, it is not well suited to some use cases.

- Many network and embedded systems, such as routers and firewalls, do not support installing third-party software. In this case it would not be possible to install the NXLog agent.
- Installing the NXLog agent on each system in a large-scale deployment may not be practical compared to reading from the existing logging daemon on each system.

3.3.2. Agent-Less Collection

With this mode of log collection, a server or device sends log data to an NXLog instance over the network, using its native protocols. NXLog collects and processes the information that it receives.

NOTE

We recommend agent-less log collection in cases where agent-based log collection is not feasible, for example from legacy or embedded systems that do not support installing the NXLog agent.

Agent-less log collection has the following advantages.

- It is not necessary to install an NXLog agent application on the target system to collect log data from it.
- Generally, a device or system requires only minimal configuration to send log data over the network to an NXLog instance in its native format.

Agent-less log collection has some disadvantages that you should consider before deploying it.

- Agent-less log collection may provide lower performance than agent-based collection. On Windows systems, the Windows Management Instrumentation process can consume more system resources than the NXLog agent.
- Reliability is also a potential issue. Since most Syslog log forwarders use UDP to transfer log data, some data could be lost if the server restarts or becomes unreachable over the network. Unlike agent-based log collection, you often cannot monitor the health of the logging source.
- The security of data transferred may be lower when using agent-less collection, since most Syslog sources transfer data over unencrypted UDP.

Agent-less collection is commonly used with the following protocols.

- BSD Syslog (RFC 3164) and IETF Syslog (RFC 5424) sources (see [Collecting and Parsing Syslog](#))
- Windows EventLog sources (with NXLog Enterprise Edition):
 - The WMI protocol, using the `im_wmi` module (see [Remote Collection With im_wmi](#))
 - The MSRPC protocol, using the `im_msvisalog` module (see [Remote Collection With im_msvisalog](#))
 - Windows Event Forwarding, using the `im_wseventing` module (see [Remote Collection With im_wseventing](#))

3.3.3. Offline Log Processing

While the other modes process log data in real-time, NXLog can also be used to perform batch log processing. This is provided by the `nxlog-processor(8)` tool, which is similar to the NXLog daemon and uses the same configuration file. However, it runs in the foreground and exits after all input log data has been processed.

Common input sources are files and databases. This tool is useful for log processing tasks such as:

- loading a group of files into a database,
- converting between different formats,
- testing patterns,
- doing offline event correlation, or
- checking HMAC message integrity.

Chapter 4. Available Modules

The following modules are provided with NXLog. Modules which are only available in NXLog Enterprise Edition are noted.

4.1. Extension Modules

The following extension (`xm_*`) modules are available.

Table 2. Available Extension Modules

Module	Description	Platforms
Remote Management (xm_admin)	Adds secure remote administration capabilities to NXLog using SOAP or JSON over HTTP/HTTPS.	All (<i>Enterprise Edition only</i>)
AIX Auditing (xm_aixaudit)	Parses AIX audit events that have been written to file.	AIX (<i>Enterprise Edition only</i>)
Apple System Logs (xm_asl)	Parses events in the Apple System Log (ASL) format.	All (<i>Enterprise Edition only</i>)
Basic Security Module Auditing (xm_bsm)	Supports parsing of events written to file in Sun's Basic Security Module (BSM) Auditing binary format.	FreeBSD, Solaris, macOS (<i>Enterprise Edition only</i>)
CEF (xm_cef)	Provides functions for generating and parsing data in the Common Event Format (CEF) used by HP ArcSight™ products.	All (<i>Enterprise Edition only</i>)
Character Set Conversion (xm_charconv)	Provides functions and procedures to help you convert strings between different character sets (code pages).	All
CSV (xm_csv)	Provides functions and procedures to help you process data formatted as comma-separated values (CSV), and to convert CSV data into fields.	All
External Program Execution (xm_exec)	Passes log data through a custom external program for processing, either synchronously or asynchronously.	All
Filelist (xm_filelist)	Implements file-based blacklisting or whitelisting.	All (<i>Enterprise Edition only</i>)
File Operations (xm_fileop)	Provides functions and procedures to manipulate files.	All
GELF (xm_gelf)	Provides an output writer function which can be used to generate output in Graylog Extended Log Format (GELF) for Graylog2 or GELF compliant tools.	All
Grok Patterns (xm_grok)	Provides support for parsing events with Grok patterns.	All (<i>Enterprise Edition only</i>)
JSON (xm_json)	Provides functions and procedures to process data that is formatted as JSON (Java Serialized Object Notation).	All
Key-Value Pairs (xm_kvp)	Provides functions and procedures to parse and generate data that is formatted as key-value pairs.	All
LEEF (xm_leef)	Provides functions for parsing and generating data in the Log Event Extended Format (LEEF), which is used by IBM Security QRadar products.	All (<i>Enterprise Edition only</i>)
DNS Server Log Parsing (xm_msdns)	Parses Microsoft Windows DNS Server logs	All (<i>Enterprise Edition only</i>)
Multi-Line Message Parser (xm_multiline)	Parses log entries that span multiple lines.	All
NetFlow (xm_netflow)	Provides a parser for NetFlow payload collected over UDP.	All (<i>Enterprise Edition only</i>)
NPS (xm_nps)	Provides functions and procedures for processing data in NPS Database Format stored in files by Microsoft Radius services.	All (<i>Enterprise Edition only</i>)

Module	Description	Platforms
Pattern Matcher (xm_pattern)	Applies advanced pattern matching logic to log data, which can give greater performance than normal regular expression statements in Exec directives. Replaces pm_pattern .	All (<i>Enterprise Edition only</i>)
Perl (xm_perl)	Processes log data using Perl.	Linux
Python (xm_python)	Processes log data using Python.	Linux (<i>Enterprise Edition only</i>)
Resolver (xm_resolver)	Resolves key identifiers that appear in log messages into more meaningful equivalents, including IP addresses to host names, and group/user IDs to friendly names.	All (<i>Enterprise Edition only</i>)
Rewrite (xm_rewrite)	Transforms event records by modifying or discarding specific fields.	All (<i>Enterprise Edition only</i>)
Ruby (xm_ruby)	Processes log data using Ruby.	Linux (<i>Enterprise Edition only</i>)
SNMP Traps (xm_snmp)	Parses SNMPv1 and SNMPv2c trap messages.	All (<i>Enterprise Edition only</i>)
Remote Management (xm_soapadmin)	Adds secure remote administration capabilities to NXLog using SOAP (web services) over HTTP/HTTPS.	All (<i>Enterprise Edition only</i>)
Syslog (xm_syslog)	Provides helpers that let you parse and output the BSD Syslog protocol as defined by RFC 3164 .	All
W3C (xm_w3c)	Parses data in the W3C Extended Log File Format, the BRO format, and Microsoft Exchange Message Tracking logs.	All (<i>Enterprise Edition only</i>)
WTMP (xm_wtmp)	Provides a parser function to process binary wtmp files.	Linux
XML (xm_xml)	Provides functions and procedures to process data that is formatted as XML.	All

4.2. Input Modules

The following input ([im_*](#)) modules are available.

Table 3. Available Input Modules

Module	Description	Platforms
BSD/Linux Process Accounting (im_acct)	Collects process accounting logs from a Linux or BSD kernel.	Linux, AIX, *BSD, Solaris, macOS
AIX Auditing (im_aixaudit)	Collects AIX audit events directly from the kernel.	AIX (<i>Enterprise Edition only</i>)
Azure (im_azure)	Collects logs from Microsoft Azure applications.	All (<i>Enterprise Edition only</i>)
Batched Compression over TCP or SSL (im_batchcompress)	Provides a compressed network transport for incoming messages with optional TLS/SSL encryption. Pairs with the om_batchcompress output module.	All (<i>Enterprise Edition only</i>)
Basic Security Module Auditing (im_bsm)	Collects audit events directly from the kernel using Sun's Basic Security Module (BSM) Auditing API.	FreeBSD, Solaris, macOS (<i>Enterprise Edition only</i>)
CheckPoint OPSEC (im_checkpoint)	Provides support for collecting logs remotely from CheckPoint devices over the OPSEC LEA protocol.	Linux (<i>Enterprise Edition only</i>)
DBI (im_dbi)	Collects log data by reading data from an SQL database using the libdbi library.	Linux
Event Tracing for Windows (ETW) (im_etw)	Implements ETW controller and consumer functionality in order to collect events from the ETW system.	Windows (<i>Enterprise Edition only</i>)

Module	Description	Platforms
Program (im_exec)	Collects log data by executing a custom external program. The standard output of the command forms the log data.	All
File (im_file)	Collects log data from a file on the local file system.	All
File Integrity Monitoring (im_fim)	Scans files and directories and reports detected changes.	All (<i>Enterprise Edition only</i>)
HTTP(S) (im_http)	Accepts incoming HTTP or HTTPS connections and collects log events from client POST requests.	All (<i>Enterprise Edition only</i>)
Internal (im_internal)	Collect log messages from NXLog.	All
Apache Kafka (im_kafka)	Implements a consumer for collecting from a Kafka cluster.	Linux, Windows (<i>Enterprise Edition only</i>)
Kernel (im_kernel)	Collects log data from the kernel log buffer.	Linux, *BSD, macOS (*BSD and macOS support in <i>Enterprise Edition only</i>)
Linux Audit System (im_linuxaudit)	Configures and collects events from the Linux Audit System	Linux (<i>Enterprise Edition only</i>)
Mark (im_mark)	Outputs 'boilerplate' log data periodically to indicate that the logger is still running.	All
MS EventLog for Windows XP/2000/2003 (im_mseventlog)	Collects EventLog messages on the Windows platform.	Windows
MS EventLog for Windows 2008/Vista and later (im_msvisalog)	Collects EventLog messages on the Windows platform.	Windows
Null (im_null)	Acts as a dummy log input module, which generates no log data. You can use this for testing purposes.	All
OCI (im_oci)	Reads log messages from an Oracle database.	Linux (<i>Enterprise Edition only</i>)
ODBC (im_odbc)	Uses the ODBC API to read log messages from database tables.	Linux, Windows (<i>Enterprise Edition only</i>)
Perl (im_perl)	Captures event data directly into NXLog using Perl code.	Linux (<i>Enterprise Edition only</i>)
Python (im_python)	Captures event data directly into NXLog using Python code.	Linux (<i>Enterprise Edition only</i>)
Redis (im_redis)	Retrieves data stored in a Redis server.	All (<i>Enterprise Edition only</i>)
Ruby (im_ruby)	Captures event data directly into NXLog using Ruby code.	Linux (<i>Enterprise Edition only</i>)
Windows Registry Monitoring (im_regmon)	Periodically scans the Windows registry and generates event records if a change in the monitored registry entries is detected.	Windows (<i>Enterprise Edition only</i>)
TLS/SSL (im_ssl)	Collects log data over a TCP connection that is secured with Transport Layer Security (TLS) or Secure Sockets Layer (SSL).	All
TCP (im_tcp)	Collects log data over a TCP network connection.	All
UDP (im_udp)	Collects log data over a UDP network connection.	All
Unix Domain Socket (im_uds)	Collects log data over a Unix domain socket (typically <code>/dev/log</code>).	Linux, AIX, *BSD, Solaris, macOS
Windows Performance Counters (im_winperfcount)	Periodically retrieves the values of the specified Windows Performance Counters to create an event record.	Windows (<i>Enterprise Edition only</i>)

Module	Description	Platforms
Windows Management Instrumentation (im_wmi)	Collects EventLog messages from Windows platforms supporting WMI mode.	Linux (<i>Enterprise Edition only</i>)
Windows Event Forwarding (im_wseventing)	Collects EventLog from Windows clients that have Windows Event Forwarding configured.	All (<i>Enterprise Edition only</i>)
ZeroMQ (im_zmq)	Provides incoming message transport over ZeroMQ, a scalable high-throughput messaging library.	Linux (<i>Enterprise Edition only</i>)

4.3. Processor Modules

The following processor ([pm_*](#)) modules are available.

Table 4. Available Processor Modules

Module	Description	Platforms
Blocker (pm_blocker)	Blocks log data from progressing through a route. You can use this module for testing purposes, to simulate when a route is blocked.	All
Buffer (pm_buffer)	Caches messages in an in-memory or disk-based buffer before forwarding it. This module is useful in combination with UDP data inputs.	All
Event Correlator (pm_evcorr)	Perform log actions based on relationships between events.	All
Filter (pm_filter)	Fowards the log data only if the condition specified in the Filter module configuration evaluates to true. This module is obsolete. Use the NXLog language drop() procedure instead.	All
HMAC Message Integrity (pm_hmac)	Protect messages with HMAC cryptographic checksumming.	All (<i>Enterprise Edition only</i>)
HMAC Message Integrity Checker (pm_hmac_check)	Check HMAC cryptographic checksums on messages.	All (<i>Enterprise Edition only</i>)
Message De-Duplicator (pm_norepeat)	Drops messages that are identical to previously-received messages.	All
Null (pm_null)	Acts as a dummy log processing module, which does not transform the log data in any way. You can use this module for testing purposes.	All
Pattern Matcher (pm_pattern)	Applies advanced pattern matching logic to log data, which can give greater performance than normal regular expression statements in Exec directives.	All
Message Format Converter (pm_transformer)	Provides parsers for various log formats, and converters between them. This module is obsolete. Use the xm_syslog, xm_csv, xm_json and xm_xml modules instead.	All
Timestamping (pm_ts)	Add cryptographic Time-Stamp signatures to messages.	All (<i>Enterprise Edition only</i>)

4.4. Output Modules

The following output ([om_*](#)) modules are available.

Table 5. Available Output Modules

Module	Description	Platforms
Batched Compression over TCP or SSL (om_batchcompress)	Provides a compressed network transport for outgoing messages with optional TLS/SSL encryption. Pairs with the im_batchcompress input module.	All (<i>Enterprise Edition only</i>)
Blocker (om_blocker)	Blocks log data from being written. You can use this module for testing purposes, to simulate when a route is blocked.	All
DBI (om_dbi)	Stores log data in an SQL database using the libdbi library.	Linux
Elasticsearch (om_elasticsearch)	Stores logs in an Elasticsearch server.	All (<i>Enterprise Edition only</i>)
EventDB (om_eventdb)	Uses libdrizzle to insert log message data into a MySQL database with a special schema.	Linux (<i>Enterprise Edition only</i>)
Program (om_exec)	Writes log data to the standard input of a custom external program.	All
File (om_file)	Writes log data to a file on the file system.	All
HTTP(s) (om_http)	Send events over HTTP or HTTPS using POST requests.	All
Apache Kafka (om_kafka)	Implements a producer for publishing to a Kafka cluster.	Linux, Windows (<i>Enterprise Edition only</i>)
Null (om_null)	Acts as a dummy log output module. The output is not written or sent anywhere. You can use this module for testing purposes.	All
OCI (om_oci)	Writes log messages to an Oracle database.	Linux (<i>Enterprise Edition only</i>)
ODBC (om_odbc)	Uses the ODBC API to write log messages to database tables.	Linux, Windows (<i>Enterprise Edition only</i>)
Perl (om_perl)	Uses Perl code to handle output log messages from NXLog.	Linux (<i>Enterprise Edition only</i>)
Python (om_python)	Uses Python code to handle output log messages from NXLog.	Linux (<i>Enterprise Edition only</i>)
Redis (om_redis)	Stores log messages in a Redis server.	All (<i>Enterprise Edition only</i>)
Ruby (om_ruby)	Uses Ruby code to handle output log messages from NXLog.	Linux (<i>Enterprise Edition only</i>)
TLS/SSL (om_ssl)	Sends log data over a TCP connection that is secured with Transport Layer Security (TLS) or Secure Sockets Layer (SSL).	All
TCP (om_tcp)	Sends log data over a TCP connection to a remote host.	All
UDP (om_udp)	Sends log data over a UDP connection to a remote host.	All
UDP with IP Spoofing (om_udpspoof)	Sends log data over a UDP connection, and spoofs the source IP address to make packets appear as if they were sent from another host.	All (<i>Enterprise Edition only</i>)
UDS (om_uds)	Sends log data to a Unix domain socket.	Linux, AIX, *BSD, Solaris, macOS
WebHDFS (om_webhdfs)	Stores log data in Hadoop HDFS using the WebHDFS protocol.	All (<i>Enterprise Edition only</i>)
ZeroMQ (om_zmq)	Provides outgoing message transport over ZeroMQ, a scalable high-throughput messaging library.	Linux (<i>Enterprise Edition only</i>)

4.5. Modules by Platform

4.5.1. AIX 6.1

Table 6. Available Modules in *nxlog-4.0.3550-1.aix6.1.ppc.rpm*

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-1.aix6.1.ppc.rpm	im_acct im_aixaudit im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.2. CentOS 6, RHEL 6

Table 7. Available Modules in *nxlog-4.0.3550_RHEL6_x86_64.tar.bz2*

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-1_rhel6.x86_64.rpm	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udpspoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint-4.0.3550-1_rhel6.x86_64.rpm	im_checkpoint			
nxlog-dbi-4.0.3550-1_rhel6.x86_64.rpm	im_dbi	om_dbi		
nxlog-odbc-4.0.3550-1_rhel6.x86_64.rpm	im_odbc	om_odbc		
nxlog-perl-4.0.3550-1_rhel6.x86_64.rpm	im_perl	om_perl		xm_perl
nxlog-wmi-4.0.3550-1_rhel6.x86_64.rpm	im_wmi			
nxlog-wseventing-4.0.3550-1_rhel6.x86_64.rpm	im_wseventing			

4.5.3. CentOS 7, RHEL 7

Table 8. Available Modules in nxlog-4.0.3550_RHEL7_x86_64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-1_rhel7.x86_64.rpm	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcom press om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udpspoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint-4.0.3550-1_rhel7.x86_64.rpm	im_checkpoint			
nxlog-dbi-4.0.3550-1_rhel7.x86_64.rpm	im_dbi	om_dbi		
nxlog-kafka-4.0.3550-1_rhel7.x86_64.rpm	im_kafka	om_kafka		
nxlog-odbc-4.0.3550-1_rhel7.x86_64.rpm	im_odbc	om_odbc		
nxlog-perl-4.0.3550-1_rhel7.x86_64.rpm	im_perl	om_perl		xm_perl
nxlog-python-4.0.3550-1_rhel7.x86_64.rpm	im_python	om_python		xm_python
nxlog-ruby-4.0.3550-1_rhel7.x86_64.rpm	im_ruby	om_ruby		xm_ruby
nxlog-wmi-4.0.3550-1_rhel7.x86_64.rpm	im_wmi			
nxlog-wseventing-4.0.3550-1_rhel7.x86_64.rpm	im_wseventing			
nxlog-zmq-4.0.3550-1_rhel7.x86_64.rpm	im_zmq	om_zmq		

4.5.4. DEB Generic

Table 9. Available Modules in nxlog-4.0.3550_generic_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml

4.5.5. Debian Jessie

Table 10. Available Modules in nxlog-4.0.3550_debian-jessie_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint_4.0.3550_amd64.deb	im_checkpoint			
nxlog-dbi_4.0.3550_amd64.deb	im_dbi	om_dbi		

Package	Input	Output	Processor	Extension
nxlog-kafka_4.0.3550_amd64.deb	im_kafka	om_kafka		
nxlog-odbc_4.0.3550_amd64.deb	im_odbc	om_odbc		
nxlog-perl_4.0.3550_amd64.deb	im_perl	om_perl		xm_perl
nxlog-python_4.0.3550_amd64.deb	im_python	om_python		xm_python
nxlog-ruby_4.0.3550_amd64.deb	im_ruby	om_ruby		xm_ruby
nxlog-wmi_4.0.3550_amd64.deb	im_wmi			
nxlog-wseventing_4.0.3550_amd64.deb	im_wseventing			
nxlog-zmq_4.0.3550_amd64.deb	im_zmq	om_zmq		

4.5.6. Debian Stretch

Table 11. Available Modules in nxlog-4.0.3550_debian-stretch_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_udns	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udpspoof om_udns om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint_4.0.3550_amd64.deb	im_checkpoint			
nxlog-dbi_4.0.3550_amd64.deb	im_dbi	om_dbi		
nxlog-kafka_4.0.3550_amd64.deb	im_kafka	om_kafka		
nxlog-odbc_4.0.3550_amd64.deb	im_odbc	om_odbc		
nxlog-perl_4.0.3550_amd64.deb	im_perl	om_perl		xm_perl
nxlog-python_4.0.3550_amd64.deb	im_python	om_python		xm_python
nxlog-ruby_4.0.3550_amd64.deb	im_ruby	om_ruby		xm_ruby
nxlog-wseventing_4.0.3550_amd64.deb	im_wseventing			
nxlog-zmq_4.0.3550_amd64.deb	im_zmq	om_zmq		

4.5.7. Debian Wheezy

Table 12. Available Modules in nxlog-4.0.3550_debian-wheezy_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcom press om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udpspoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint_4.0.3550_amd64.deb	im_checkpoint			
nxlog-dbi_4.0.3550_amd64.deb	im_dbi	om_dbi		
nxlog-odbc_4.0.3550_amd64.deb	im_odbc	om_odbc		
nxlog-perl_4.0.3550_amd64.deb	im_perl	om_perl		xm_perl
nxlog-python_4.0.3550_amd64.deb	im_python	om_python		xm_python
nxlog-wmi_4.0.3550_amd64.deb	im_wmi			
nxlog-wseventing_4.0.3550_amd64.deb	im_wseventing			

4.5.8. FreeBSD 11

Table 13. Available Modules in nxlog-4.0.3550-fbsd.tgz

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-fbsd.tgz	im_acct im_azure im_batchcompress im_bsm im_exec im_file im_fim im_http im_internal im_kernel im_mark im_null im_python im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_python om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_check pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_bsm xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_python xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.9. MacOS

Table 14. Available Modules in nxlog-4.0.3550-macos-x86.pkg

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-macos-x86.pkg	im_acct im_azure im_batchcompress im_bsm im_exec im_file im_fim im_http im_internal im_kernel im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_check pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_bsm xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_python xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.10. Microsoft Windows 32bit

Table 15. Available Modules in nxlog-4.0.3550-x86.msi

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-x86.msi	im_azure im_batchcompress im_etw im_exec im_file im_fim im_http im_internal im_mark im_mseventlog im_msvisalog im_null im_odbc im_perl im_redis im_regmon im_ssl im_tcp im_testgen im_udp im_winperfcount im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_odbc om_perl om_redis om_ssl om_tcp om_udp om_udpspoof om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_perl xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.11. Microsoft Windows 64bit

Table 16. Available Modules in nxlog-4.0.3550-x64.msi

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-x64.msi	im_azure im_batchcompress im_etw im_exec im_file im_fim im_http im_internal im_kafka im_mark im_mseventlog im_msvisalog im_null im_odbc im_perl im_redis im_regmon im_ssl im_tcp im_testgen im_udp im_winperfcount im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_kafka om_null om_odbc om_perl om_redis om_ssl om_tcp om_udp om_udpspoof om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_perl xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.12. OpenBSD 6.0

Table 17. Available Modules in nxlog-4.0.3550-obsd6_0.tgz

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-obsd6_0.tgz	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_mark im_null im_python im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_python om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_checker pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_python xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.13. OpenBSD 6.2

Table 18. Available Modules in nxlog-4.0.3550-obsd6_2.tgz

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-obsd6_2.tgz	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_python om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_checker pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.14. RPM Generic

Table 19. Available Modules in nxlog-4.0.3550_GENERIC_x86_64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-1_generic.x86_64.rpm	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_check pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml

4.5.15. SLES 11

Table 20. Available Modules in nxlog-4.0.3550_SLES11_x86_64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-1_sles11.x86_64.rpm	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_check pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml

Package	Input	Output	Processor	Extension
nxlog-odbc-4.0.3550-1_sles11.x86_64.rpm	im_odb	om_odb		
nxlog-perl-4.0.3550-1_sles11.x86_64.rpm	im_perl	om_perl		xm_perl
nxlog-wseventing-4.0.3550-1_sles11.x86_64.rpm	im_wseventing			

4.5.16. SLES 12

Table 21. Available Modules in nxlog-4.0.3550_SLES12_x86_64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog-4.0.3550-1_sles12.x86_64.rpm	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcompress om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-dbi-4.0.3550-1_sles12.x86_64.rpm	im_dbi	om_dbi		
nxlog-odbc-4.0.3550-1_sles12.x86_64.rpm	im_odb	om_odb		
nxlog-perl-4.0.3550-1_sles12.x86_64.rpm	im_perl	om_perl		xm_perl
nxlog-python-4.0.3550-1_sles12.x86_64.rpm	im_python	om_python		xm_python
nxlog-wmi-4.0.3550-1_sles12.x86_64.rpm	im_wmi			
nxlog-wseventing-4.0.3550-1_sles12.x86_64.rpm	im_wseventing			
nxlog-zmq-4.0.3550-1_sles12.x86_64.rpm	im_zmq	om_zmq		

4.5.17. Solaris 10 i386

Table 22. Available Modules in nxlog-4.0.3550.solaris-x86.pkg.gz

Package	Input	Output	Processor	Extension
nxlog-4.0.3550.solaris-x86.pkg.gz	im_acct im_azure im_batchcompress im_bsm im_exec im_file im_fim im_http im_internal im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_bsm xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.18. Solaris 10 sparc

Table 23. Available Modules in nxlog-4.0.3550.solaris-sparc.pkg.gz

Package	Input	Output	Processor	Extension
nxlog-4.0.3550.solaris-sparc.pkg.gz	im_acct im_azure im_batchcompress im_bsm im_exec im_file im_fim im_http im_internal im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds im_wseventing	om_batchcompress om_blocker om_elasticsearch om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udsp spoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_bsm xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kvp xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_xml

4.5.19. Ubuntu 12.04

Table 24. Available Modules in nxlog-4.0.3550_ubuntu-precise_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcom press om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udspoo om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ch ck pm_hmac pm_norepeat pm_null pm_pattern pm_transform er pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmi n xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint_4.0.3550_amd64.deb	im_checkpoint			
nxlog-dbi_4.0.3550_amd64.deb	im_dbi	om_dbi		
nxlog-odbc_4.0.3550_amd64.deb	im_odbc	om_odbc		
nxlog-perl_4.0.3550_amd64.deb	im_perl	om_perl		xm_perl
nxlog-wmi_4.0.3550_amd64.deb	im_wmi			
nxlog-wseventing_4.0.3550_amd64.deb	im_wseventing			

4.5.20. Ubuntu 14.04

Table 25. Available Modules in nxlog-4.0.3550_ubuntu-trusty_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcom press om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udpspoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint_4.0.3550_amd64.deb	im_checkpoint			
nxlog-dbi_4.0.3550_amd64.deb	im_dbi	om_dbi		
nxlog-kafka_4.0.3550_amd64.deb	im_kafka	om_kafka		
nxlog-odbc_4.0.3550_amd64.deb	im_odbc	om_odbc		
nxlog-perl_4.0.3550_amd64.deb	im_perl	om_perl		xm_perl
nxlog-python_4.0.3550_amd64.deb	im_python	om_python		xm_python
nxlog-wmi_4.0.3550_amd64.deb	im_wmi			
nxlog-wseventing_4.0.3550_amd64.deb	im_wseventing			
nxlog-zmq_4.0.3550_amd64.deb	im_zmq	om_zmq		

4.5.21. Ubuntu 16.04

Table 26. Available Modules in nxlog-4.0.3550_ubuntu-xenial_amd64.tar.bz2

Package	Input	Output	Processor	Extension
nxlog_4.0.3550_amd64.deb	im_acct im_azure im_batchcompress im_exec im_file im_fim im_http im_internal im_kernel im_linuxaudit im_mark im_null im_redis im_ssl im_tcp im_testgen im_udp im_uds	om_batchcom press om_blocker om_elasticsearch om_eventdb om_exec om_file om_http om_null om_redis om_ssl om_tcp om_udp om_udpspoof om_uds om_webhdfs	pm_blocker pm_buffer pm_evcorr pm_filter pm_hmac_ck pm_hmac pm_norepeat pm_null pm_pattern pm_transformer pm_ts	xm_admin xm_aixaudit xm_asl xm_cef xm_charconv xm_csv xm_exec xm_filelist xm_fileop xm_gelf xm_grok xm_json xm_kv xm_leef xm_msdns xm_multiline xm_netflow xm_nps xm_pattern xm_resolver xm_rewrite xm_snmp xm_soapadmin xm_stdinpw xm_syslog xm_w3c xm_wtmp xm_xml
nxlog-checkpoint_4.0.3550_amd64.deb	im_checkpoint			
nxlog-dbi_4.0.3550_amd64.deb	im_dbi	om_dbi		
nxlog-kafka_4.0.3550_amd64.deb	im_kafka	om_kafka		
nxlog-odbc_4.0.3550_amd64.deb	im_odbc	om_odbc		
nxlog-perl_4.0.3550_amd64.deb	im_perl	om_perl		xm_perl
nxlog-python_4.0.3550_amd64.deb	im_python	om_python		xm_python
nxlog-ruby_4.0.3550_amd64.deb	im_ruby	om_ruby		xm_ruby
nxlog-wseventing_4.0.3550_amd64.deb	im_wseventing			
nxlog-zmq_4.0.3550_amd64.deb	im_zmq	om_zmq		

4.6. Modules by Package

Table 27. Input Modules

im_acct	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_aixaudit	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1)
im_azure	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_batchcompress	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_bsm	nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc)

im_checkpoint	nxlog-checkpoint-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-checkpoint-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-checkpoint_4.0.3550_amd64.deb (Debian Jessie) nxlog-checkpoint_4.0.3550_amd64.deb (Debian Stretch) nxlog-checkpoint_4.0.3550_amd64.deb (Debian Wheezy) nxlog-checkpoint_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-checkpoint_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-checkpoint_4.0.3550_amd64.deb (Ubuntu 16.04)
im_dbi	nxlog-dbi-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-dbi-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-dbi_4.0.3550_amd64.deb (Debian Jessie) nxlog-dbi_4.0.3550_amd64.deb (Debian Stretch) nxlog-dbi_4.0.3550_amd64.deb (Debian Wheezy) nxlog-dbi-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-dbi_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-dbi_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-dbi_4.0.3550_amd64.deb (Ubuntu 16.04)
im_etw	nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit)
im_exec	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_file	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

im_fim	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_http	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_internal	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_kafka	nxlog-kafka-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-kafka_4.0.3550_amd64.deb (Debian Jessie) nxlog-kafka_4.0.3550_amd64.deb (Debian Stretch) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-kafka_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-kafka_4.0.3550_amd64.deb (Ubuntu 16.04)

im_kernel	nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_linuxaudit	nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_mark	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_mseventlog	nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit)
im_msvisatalog	nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit)

im_null	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_odbc	nxlog-odbc-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-odbc-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-odbc_4.0.3550_amd64.deb (Debian Jessie) nxlog-odbc_4.0.3550_amd64.deb (Debian Stretch) nxlog-odbc_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-odbc-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-odbc-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-odbc_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-odbc_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-odbc_4.0.3550_amd64.deb (Ubuntu 16.04)
im_perl	nxlog-perl-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-perl-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-perl_4.0.3550_amd64.deb (Debian Jessie) nxlog-perl_4.0.3550_amd64.deb (Debian Stretch) nxlog-perl_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-perl-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-perl-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 16.04)
im_python	nxlog-python-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-python_4.0.3550_amd64.deb (Debian Jessie) nxlog-python_4.0.3550_amd64.deb (Debian Stretch) nxlog-python_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-python-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-python_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-python_4.0.3550_amd64.deb (Ubuntu 16.04)

im_redis	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_regmon	nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit)
im_ruby	nxlog-ruby-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-ruby_4.0.3550_amd64.deb (Debian Jessie) nxlog-ruby_4.0.3550_amd64.deb (Debian Stretch) nxlog-ruby_4.0.3550_amd64.deb (Ubuntu 16.04)
im_ssl	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_tcp	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

im_testgen	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_udp	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_uds	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
im_winperfcount	nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit)
im_wmi	nxlog-wmi-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-wmi-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-wmi_4.0.3550_amd64.deb (Debian Jessie) nxlog-wmi_4.0.3550_amd64.deb (Debian Wheezy) nxlog-wmi-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-wmi_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-wmi_4.0.3550_amd64.deb (Ubuntu 14.04)

im_wseventing	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-wseventing-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-wseventing-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog-wseventing_4.0.3550_amd64.deb (Debian Jessie) nxlog-wseventing_4.0.3550_amd64.deb (Debian Stretch) nxlog-wseventing_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-wseventing-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-wseventing-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog-wseventing_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-wseventing_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-wseventing_4.0.3550_amd64.deb (Ubuntu 16.04)
im_zmq	nxlog-zmq-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-zmq_4.0.3550_amd64.deb (Debian Jessie) nxlog-zmq_4.0.3550_amd64.deb (Debian Stretch) nxlog-zmq-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-zmq_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-zmq_4.0.3550_amd64.deb (Ubuntu 16.04)

Table 28. Output Modules

om_batchcompress	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
------------------	---

om_blocker	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_dbi	nxlog-dbi-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-dbi-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-dbi_4.0.3550_amd64.deb (Debian Jessie) nxlog-dbi_4.0.3550_amd64.deb (Debian Stretch) nxlog-dbi_4.0.3550_amd64.deb (Debian Wheezy) nxlog-dbi-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-dbi_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-dbi_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-dbi_4.0.3550_amd64.deb (Ubuntu 16.04)
om_elasticsearch	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_eventdb	nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

om_exec	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_file	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_http	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_kafka	nxlog-kafka-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-kafka_4.0.3550_amd64.deb (Debian Jessie) nxlog-kafka_4.0.3550_amd64.deb (Debian Stretch) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-kafka_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-kafka_4.0.3550_amd64.deb (Ubuntu 16.04)

om_null	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_odbc	nxlog-odbc-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-odbc-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-odbc_4.0.3550_amd64.deb (Debian Jessie) nxlog-odbc_4.0.3550_amd64.deb (Debian Stretch) nxlog-odbc_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-odbc-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-odbc-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-odbc_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-odbc_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-odbc_4.0.3550_amd64.deb (Ubuntu 16.04)
om_perl	nxlog-perl-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-perl-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-perl_4.0.3550_amd64.deb (Debian Jessie) nxlog-perl_4.0.3550_amd64.deb (Debian Stretch) nxlog-perl_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-perl-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-perl-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 16.04)
om_python	nxlog-python-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-python_4.0.3550_amd64.deb (Debian Jessie) nxlog-python_4.0.3550_amd64.deb (Debian Stretch) nxlog-python_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-python-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-python_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-python_4.0.3550_amd64.deb (Ubuntu 16.04)

om_redis	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_ruby	nxlog-ruby-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-ruby_4.0.3550_amd64.deb (Debian Jessie) nxlog-ruby_4.0.3550_amd64.deb (Debian Stretch) nxlog-ruby_4.0.3550_amd64.deb (Ubuntu 16.04)
om_ssl	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_tcp	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

om_udp	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_udpspoof	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_uds	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

om_webhdfs	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
om_zmq	nxlog-zmq-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-zmq_4.0.3550_amd64.deb (Debian Jessie) nxlog-zmq_4.0.3550_amd64.deb (Debian Stretch) nxlog-zmq-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-zmq_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-zmq_4.0.3550_amd64.deb (Ubuntu 16.04)

Table 29. Extension Modules

xm_admin	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
----------	---

xm_aixaudit	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_asl	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_bsm	nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc)
xm_cef	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_charconv	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_csv	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_exec	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_filelist	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_fileop	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_gelf	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_grok	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_json	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_kvp	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_leef	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_msdns	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_multiline	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_netflow	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_nps	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_pattern	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_perl	nxlog-perl-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-perl-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-perl_4.0.3550_amd64.deb (Debian Jessie) nxlog-perl_4.0.3550_amd64.deb (Debian Stretch) nxlog-perl_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-perl-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-perl-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-perl_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_python	nxlog-python-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-python_4.0.3550_amd64.deb (Debian Jessie) nxlog-python_4.0.3550_amd64.deb (Debian Stretch) nxlog-python_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-python-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-python_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog-python_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_resolver	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_rewrite	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-obsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-obsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_ruby	nxlog-ruby-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog-ruby_4.0.3550_amd64.deb (Debian Jessie) nxlog-ruby_4.0.3550_amd64.deb (Debian Stretch) nxlog-ruby_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_snmp	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_soapadmin	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_stdinpw	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_syslog	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_w3c	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
xm_wtmp	nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

xm_xml	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
--------	---

Table 30. Processor Modules

pm_blocker	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_buffer	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

pm_evcoll	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_filter	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_hmac	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

pm_hmac_check	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_norepeat	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_null	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

pm_pattern	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_transformer	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)
pm_ts	nxlog-4.0.3550-1.aix6.1.ppc.rpm (AIX 6.1) nxlog-4.0.3550-1_rhel6.x86_64.rpm (CentOS 6, RHEL 6) nxlog-4.0.3550-1_rhel7.x86_64.rpm (CentOS 7, RHEL 7) nxlog_4.0.3550_amd64.deb (DEB Generic) nxlog_4.0.3550_amd64.deb (Debian Jessie) nxlog_4.0.3550_amd64.deb (Debian Stretch) nxlog_4.0.3550_amd64.deb (Debian Wheezy) nxlog-4.0.3550-fbsd.tgz (FreeBSD 11) nxlog-4.0.3550-macos-x86.pkg (MacOS) nxlog-4.0.3550-x86.msi (Microsoft Windows 32bit) nxlog-4.0.3550-x64.msi (Microsoft Windows 64bit) nxlog-4.0.3550-bsd6_0.tgz (OpenBSD 6.0) nxlog-4.0.3550-bsd6_2.tgz (OpenBSD 6.2) nxlog-4.0.3550-1_generic.x86_64.rpm (RPM Generic) nxlog-4.0.3550-1_sles11.x86_64.rpm (SLES 11) nxlog-4.0.3550-1_sles12.x86_64.rpm (SLES 12) nxlog-4.0.3550.solaris-x86.pkg.gz (Solaris 10 i386) nxlog-4.0.3550.solaris-sparc.pkg.gz (Solaris 10 sparc) nxlog_4.0.3550_amd64.deb (Ubuntu 12.04) nxlog_4.0.3550_amd64.deb (Ubuntu 14.04) nxlog_4.0.3550_amd64.deb (Ubuntu 16.04)

Deployment

Chapter 5. Supported Platforms

The following operating systems and architectures are fully supported, except as noted.

Table 31. Supported GNU/Linux Platforms

Operating System	Architectures
RedHat Enterprise Linux 6	x86 (see note), x86_64
RedHat Enterprise Linux 7	x86 (see note), x86_64
CentOS Linux 6	x86 (see note), x86_64
CentOS Linux 7	x86 (see note), x86_64
Debian GNU/Linux 7 (Wheezy)	x86 (see note), x86_64
Debian GNU/Linux 8 (Jessie)	x86 (see note), x86_64
Debian GNU/Linux 9 (Stretch)	x86 (see note), x86_64
Ubuntu 12.04 (Precise Pangolin)	x86 (see note), x86_64
Ubuntu 14.04 (Trusty Tahr)	x86 (see note), x86_64
Ubuntu 16.04 (Xenial Xerus)	x86 (see note), x86_64
SUSE Linux Enterprise Server 11	x86 (see note), x86_64
SUSE Linux Enterprise Server 12	x86 (see note), x86_64
Other distributions	(see note)

Table 32. Supported BSD Platforms

Operating System	Architectures
FreeBSD 10	x86 (see note), x86_64
FreeBSD 11	x86 (see note), x86_64
OpenBSD 5.9	x86 (see note), x86_64
OpenBSD 6.0	x86 (see note), x86_64

NOTE

Under the *Technical Support Services Agreement*, Linux and BSD binary packages may be provided upon request for operating systems that have reached their end-of-life date (like RHEL 5), for legacy 32-bit hardware, or for less common distributions (such as Linux Mint).

Table 33. Supported Microsoft Windows Platforms

Operating System	Architectures
Microsoft Windows Server 2008	x86, x86_64
Microsoft Windows Server 2012	x86_64
Microsoft Windows Server 2016	x86_64
Microsoft Windows Vista	x86, x86_64
Microsoft Windows 7	x86, x86_64
Microsoft Windows 8	x86, x86_64
Microsoft Windows 10	x86, x86_64

The following Microsoft Windows operating systems are unsupported due to having reached end-of-life status, but are known to work with NXLog.

Table 34. End-of-Life Windows Platforms

Operating System	Architectures
Microsoft Windows XP	x86, x86_64
Microsoft Windows Server 2000	x86
Microsoft Windows Server 2003	x86, x86_64

Table 35. Supported Oracle Solaris Platforms

Operating System	Architectures
Oracle Solaris 10	x86, SPARC
Oracle Solaris 11	x86, SPARC

Table 36. Supported Apple macOS Platforms

Operating System	Architectures
OS X 10.11 (El Capitan)	x86_64
macOS 10.12 (Sierra)	x86_64

Table 37. Supported IBM AIX Platforms

Operating System	Architectures
AIX 6.1	PowerPC
AIX 7.1	PowerPC

Chapter 6. System Requirements

In order to function efficiently, each NXLog product requires a certain amount of available system resources on the host system. The tables below provide general guidelines to use when planning an NXLog deployment. Actual system requirements will vary based on the configuration and event rate; therefore, both minimum and recommended requirements are listed. Always thoroughly test a deployment to verify that the desired performance can be achieved with the system resources available.

NOTE

These requirements are in addition to the operating system's requirements, and the requirements should be combined cumulatively for systems running both NXLog Enterprise Edition and NXLog Manager.

Table 38. NXLog Enterprise Edition Requirements

	Minimum	Recommended
Processor cores	1	2
Memory/RAM	60 MB	250 MB
Disk space	50 MB	150 MB

Table 39. NXLog Manager Requirements

	Minimum	Recommended
Processor cores	2	2
Memory/RAM (see note)	2048 MB	4096 MB
Disk space	300 MB	1024 MB

NOTE

The NXLog Manager memory/RAM requirement increases by 2 MB for each managed agent. For example, if an NXLog Manager instance monitors 100 agents, the recommended memory/RAM requirement is 4296 MB.

Chapter 7. Red Hat Enterprise Linux & CentOS

7.1. Installing

1. Download the appropriate NXLog install archive from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the correct archive for your system.

Table 40. Available RHEL/CentOS Files

Platform	Archive
RHEL 6 or CentOS 6	nxlog-4.0.3550_rhel6.x86_64.tar.bz2
RHEL 7 or CentOS 7	nxlog-4.0.3550_rhel7.x86_64.tar.bz2
Generic RPM	nxlog-4.0.3550_generic.x86_64.rpm

The RHEL 6 and RHEL 7 archives above each contain several RPMs (see [Packages in a RHEL Archive](#) below). These RPMs have dependencies on system-provided RPMs.

NOTE The generic RPM above contains all the libraries (such as libpcre and libexpat) that are needed by NXLog, the only dependency is libc. However, some modules are not available (im_checkpoint, for example). The advantage of the generic RPM is that it can be installed on most RPM-based Linux distributions.

2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Log in to the target server and extract the contents of the archive.

```
$ tar -xjf nxlog-4.0.3550_rhel7.x86_64.tar.bz2
```

Table 41. Packages in a RHEL Archive

Package	Description
nxlog-4.0.3550_rhel7.x86_64.rpm	The main NXLog package
nxlog-checkpoint-4.0.3550_rhel7.x86_64.rpm	Provides the im_checkpoint module
nxlog-dbi-4.0.3550_rhel7.x86_64.rpm	Provides the im_dbi and om_dbi modules
nxlog-odbc-4.0.3550_rhel7.x86_64.rpm	Provides the im_odbc and om_odbc modules
nxlog-perl-4.0.3550_rhel7.x86_64.rpm	Provides the xm_perl , im_perl , and om_perl modules
nxlog-wmi-4.0.3550_rhel7.x86_64.rpm	Provides the im_wmi module
nxlog-wseventing-4.0.3550_rhel7.x86_64.rpm	Provides the im_wseventing module
nxlog-zmq-4.0.3550_rhel7.x86_64.rpm	Provides the im_zmq and om_zmq modules

4. Install the required NXLog packages and their dependencies. The following example installs the main NXLog package only.

```
$ sudo yum install nxlog-4.0.3550_rhel7.x86_64.rpm
```

NOTE If you are installing the [nxlog-zmq](#) package, the EPEL repository must first be enabled with [yum install -y epel-release](#).

5. Configure NXLog by editing [/opt/nxlog/etc/nxlog.conf](#). General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on Linux, see the

GNU/Linux summary.

6. Verify the configuration file syntax.

```
$ sudo /opt/nxlog/bin/nxlog -v  
2017-03-17 08:05:06 INFO configuration OK
```

7. Start the service using the `service` command:

```
$ sudo service nxlog start
```

8. Check that the NXLog service is running with the `systemctl` command.

```
$ systemctl | grep nxlog  
nxlog.service          loaded active running    LSB: logging daemon
```

7.2. Upgrading

To update an NXLog installation to the latest release, use `yum` as in the [installation instructions](#) above.

```
$ sudo yum install nxlog-4.0.3550_rhel7.x86_64.rpm
```

To replace a trial installation of NXLog Enterprise Edition with a licensed copy of the same version, follow the [installation instructions](#) but use `yum reinstall`.

```
$ sudo yum reinstall nxlog-4.0.3550_rhel7.x86_64.rpm
```

7.3. Uninstalling

To uninstall NXLog, use `yum remove`. To remove any packages that were dependencies of NXLog but are not required by any other packages, include the `--setopt=clean_requirements_on_remove=1` option. Verify the operation before confirming!

```
$ sudo yum remove 'nxlog-*'
```

NOTE

This procedure may not remove all files that were created in order to configure NXLog, or that were created as a result of NXLog's logging operations. To find these files, consult the configuration files that were used with NXLog and check the installation directory (`/opt/nxlog`).

Chapter 8. Debian & Ubuntu

8.1. Installing

1. Download the appropriate NXLog install archive from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the correct archive for your system.

Table 42. Available Debian/Ubuntu Archives

Platform	Archive
Debian 7 (Wheezy)	nxlog-4.0.3550_debian-wheezy_amd64.tar.bz2
Debian 8 (Jessie)	nxlog-4.0.3550_debian-jessie_amd64.tar.bz2
Debian 9 (Stretch)	nxlog-4.0.3550_debian-stretch_amd64.tar.bz2
Ubuntu 12.04 (Precise Pangolin)	nxlog-4.0.3550_ubuntu-precise_amd64.tar.bz2
Ubuntu 14.04 (Trusty Tahr)	nxlog-4.0.3550_ubuntu-trusty_amd64.tar.bz2
Ubuntu 16.04 (Xenial Xerus)	nxlog-4.0.3550_ubuntu-xenial_amd64.tar.bz2

2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Log in to the target server and extract the contents of the archive.

```
$ tar -xjf nxlog-4.0.3550_debian-stretch_amd64.tar.bz2
```

Table 43. Packages in a Debian/Ubuntu Archive

Package	Description
nxlog-4.0.3550_amd64.deb	The main NXLog package
nxlog-checkpoint-4.0.3550_amd64.deb	Provides the im_checkpoint module
nxlog-dbi-4.0.3550_amd64.deb	Provides the im_dbi and om_dbi modules
nxlog-odbc-4.0.3550_amd64.deb	Provides the im_odbc and om_odbc modules
nxlog-perl-4.0.3550_amd64.deb	Provides the xm_perl , im_perl , and om_perl modules
nxlog-wmi-4.0.3550_amd64.deb	Provides the im_wmi module
nxlog-wseventing-4.0.3550_amd64.deb	Provides the im_wseventing module
nxlog-zmq-4.0.3550_amd64.deb	Provides the im_zmq and om_zmq modules

4. Install the required NXLog packages and their dependencies.
 - a. First, use dpkg to install the NXLog packages (this example installs the main NXLog package only).


```
$ sudo dpkg -i nxlog-4.0.3550_amd64.deb
```
 - b. Then, if dpkg returned errors about uninstalled dependencies, use apt-get to install them and complete the NXLog installation.


```
$ sudo apt-get -f install
```
5. Configure NXLog by editing [/opt/nxlog/etc/nxlog.conf](#). General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on Linux, see the [GNU/Linux](#) summary.
6. Verify the configuration file syntax.

```
$ sudo /opt/nxlog/bin/nxlog -v  
2017-03-17 08:05:06 INFO configuration OK
```

7. Start the service using the **service** command:

```
$ sudo service nxlog start
```

8. Check that the NXLog service is running with the **service** command.

```
$ sudo service nxlog status  
● nxlog.service - LSB: logging daemon  
  Loaded: loaded (/etc/init.d/nxlog)  
  Active: active (running) since Wed 2016-10-19 22:21:36 BST; 3h 49min ago  
    Process: 518 ExecStart=/etc/init.d/nxlog start (code=exited, status=0/SUCCESS)  
   CGroup: /system.slice/nxlog.service  
         └─6297 /opt/nxlog/bin/nxlog  
[...]
```

8.2. Upgrading

To update an NXLog installation to the latest release, or to replace a trial installation of NXLog Enterprise Edition with a licensed copy, use **dpkg** as in the [installation instructions](#) above.

```
$ sudo dpkg -i nxlog-4.0.3550_amd64.deb
```

If **dpkg** returns errors about uninstalled dependencies, resolve with **apt-get**.

```
$ sudo apt-get -f install
```

8.3. Uninstalling

To uninstall NXLog, use **apt-get**. To remove any unused dependencies (system-wide), include the **--auto-remove** option. Verify the operation before confirming!

```
$ sudo apt-get remove '^nxlog*'
```

NOTE

Use **apt-get purge** instead to also remove configuration files. But in either case, this procedure may not remove all files that were created in order to configure NXLog, or that were created as a result of NXLog's logging operations. To find these files, consult the configuration files that were used with NXLog and check the installation directory (**/opt/nxlog**).

Chapter 9. SUSE Linux Enterprise Server

9.1. Installing

1. Download the appropriate NXLog install archive from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the correct archive for your system.

Table 44. Available SLES Files

Platform	Archive
SUSE Linux Enterprise Server 11	nxlog-4.0.3550_sles11.x86_64.tar.bz2
SUSE Linux Enterprise Server 12	nxlog-4.0.3550_sles12.x86_64.tar.bz2

NOTE The SLES 11 and SLES 12 archives above each contain several RPMs (see [Packages in an SLES Archive](#) below). These RPMs have dependencies on system-provided RPMs.

2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Log in to the target server and extract the contents of the archive.

```
$ tar xjf nxlog-4.0.3550_sles12.x86_64.tar.bz2
```

Table 45. Packages in an SLES Archive

Package	Description
nxlog-4.0.3550_sles12.x86_64.rpm	The main NXLog package
nxlog-dbi-4.0.3550_sles12.x86_64.rpm	Provides the im_dbi and om_dbi modules
nxlog-odbc-4.0.3550_sles12.x86_64.rpm	Provides the im_odbc and om_odbc modules
nxlog-perl-4.0.3550_sles12.x86_64.rpm	Provides the xm_perl , im_perl , and om_perl modules
nxlog-wmi-4.0.3550_sles12.x86_64.rpm	Provides the im_wmi module
nxlog-wseventing-4.0.3550_sles12.x86_64.rpm	Provides the im_wseventing module

4. Install the required NXLog packages and their dependencies (this example installs the main NXLog package only).

```
$ sudo zypper install nxlog-4.0.3550_sles12.x86_64.rpm
```

5. Configure NXLog by editing **/opt/nxlog/etc/nxlog.conf**. General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on Linux, see the [GNU/Linux](#) summary.
6. Verify the configuration file syntax.

```
$ sudo /opt/nxlog/bin/nxlog -v
2017-03-17 08:05:06 INFO configuration OK
```

7. Start the service using the **service** command:

```
$ systemctl start nxlog.service
```

8. Check that the NXLog service is running with the **systemctl** command.

```
$ systemctl | grep nxlog  
nxlog.service          loaded active running    LSB: logging daemon
```

9.2. Upgrading

To update an NXLog installation to the latest release, use **zypper** as in the [installation instructions](#) above.

```
$ sudo zypper install nxlog-4.0.3550_sles12.x86_64.rpm
```

To replace a trial installation of NXLog Enterprise Edition with a licensed copy of the same version, follow the [installation instructions](#) but use **zypper install -f**.

```
$ sudo zypper install -f nxlog-4.0.3550_rhel7.x86_64.rpm
```

9.3. Uninstalling

To uninstall NXLog, use **zypper remove**. To remove any packages that were dependencies of NXLog but are not required by any other packages, include the **--clean-deps** option. Verify the operation before confirming!

```
$ sudo zypper remove 'nxlog*'
```

NOTE

This procedure may not remove all files that were created in order to configure NXLog, or that were created as a result of NXLog's logging operations. To find these files, consult the configuration files that were used with NXLog and check the installation directory ([/opt/nxlog](#)).

Chapter 10. FreeBSD

10.1. Installing

NXLog is available in a precompiled package for FreeBSD. Follow these steps to install NXLog.

1. Download the appropriate NXLog install archive from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the `nxlog-4.0.3550-fbsd.tgz` package.
2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Log in to the target server as the root user.
4. Install NXLog with the `pkg(7)` utility.

```
# pkg add nxlog-4.0.3550-fbsd.tgz
Installing nxlog-4.0.3550-fbsd...
Extracting nxlog-4.0.3550-fbsd: 100%
```

The installation prefix is `/opt/nxlog`. Configuration files are located in `/opt/nxlog/etc`. The `rc` init script is located in `/opt/nxlog/share/nxlog/init` and is placed in `/etc/rc.d` on installation. An `nxlog` user account is created, and NXLog will run under this user by default.

5. Copy and edit the configuration file.

```
# cp /opt/nxlog/etc/nxlog.conf.sample /opt/nxlog/etc/nxlog.conf
# vi /opt/nxlog/etc/nxlog.conf
```

General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on BSD, see the [FreeBSD and OpenBSD](#) summary.

6. Verify the configuration file syntax.

```
# /opt/nxlog/bin/nxlog -v
2017-03-17 08:05:06 INFO configuration OK
```

7. To enable NXLog, add the line `nxlog_enable="YES"` to `rc.conf`. Then manage the NXLog service with the `service(8)` utility.

```
# service nxlog start
# service nxlog status
nxlog is running as pid 83708.
# service nxlog stop
process 83708 stopped
```

10.2. Upgrading

To upgrade NXLog, first remove the old version and then install the new version.

1. Remove the installed version of NXLog with the `pkg(7)` utility.

```
# pkg delete nxlog
Checking integrity... done (0 conflicting)
Deinstallation has been requested for the following 1 packages (of 0 packages
in the universe):

Installed packages to be REMOVED:
  nxlog-4.0.3550-fbsd

Number of packages to be removed: 1

The operation will free 39 MiB.

Proceed with deinstalling packages? [y/N]: y
[1/1] Deinstalling nxlog-4.0.3550-fbsd...
[1/1] Deleting files for nxlog-4.0.3550-fbsd: 100%
```

2. Install the new version.

```
# pkg add nxlog-4.0.3550-fbsd.tgz
Installing nxlog-4.0.3550-fbsd...
Extracting nxlog-4.0.3550-fbsd: 100%
```

3. Restart the NXLog service.

```
# service nxlog restart
```

10.3. Uninstalling

1. Use the **pkg(7)** utility to uninstall the NXLog package.

```
# pkg delete nxlog
Updating database digests format: 100%
Checking integrity... done (0 conflicting)
Deinstallation has been requested for the following 1 packages (of 0 packages
in the universe):

Installed packages to be REMOVED:
  nxlog-4.0.3550-fbsd

Number of packages to be removed: 1

The operation will free 92 MiB.

Proceed with deinstalling packages? [y/N]: y
[1/1] Deinstalling nxlog-4.0.3550-fbsd...
[1/1] Deleting files for nxlog-4.0.3550-fbsd: 100%
```

The uninstall script will remove NXLog along with the user, group, and files. The **pkg** utility will *not* remove new or modified files.

2. Manually remove the base directory. This will remove any new or modified files left behind by the previous step.

```
# rm -rf /opt/nxlog
```

Chapter 11. OpenBSD

11.1. Installing

NXLog comes in precompiled packages for OpenBSD. To install NXLog take the following actions.

1. Download the appropriate NXLog install archive from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the correct package for your system.

Table 46. Available OpenBSD Packages

Platform	Package
OpenBSD 5.9	nxlog-4.0.3550-obsd5.tgz
OpenBSD 6.0	nxlog-4.0.3550-obsd6.tgz

2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Log in to the target server as the root user.
4. Install NXLog with the **pkg_add(1)** utility. The OpenBSD package is currently unsigned, use the **-D unsigned** flag to install.

```
# pkg_add -D unsigned nxlog-4.0.3550-obsd6.tgz
nxlog-4.0.3550-obsd6: ok
```

The installation prefix is **/opt/nxlog**. Configuration files are located in **/opt/nxlog/etc**. The **rc** init script is located in **/opt/nxlog/share/nxlog/init** and is placed in **/etc/rc.d** on installation.

5. Copy and edit the configuration file.

```
# cp /opt/nxlog/etc/nxlog.conf.sample /opt/nxlog/etc/nxlog.conf
# vi /opt/nxlog/etc/nxlog.conf
```

General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on BSD, see the [FreeBSD and OpenBSD](#) summary.

6. Verify the configuration file syntax.

```
# /opt/nxlog/bin/nxlog -v
2017-03-17 08:05:06 INFO configuration OK
```

7. Manage the service using the **rcctl(8)** utility.

```
# rcctl enable nxlog
# rcctl start nxlog
nxlog(ok)
# rcctl stop nxlog
nxlog(ok)
# rcctl disable nxlog
```

You can also use **rcctl(8)** to check and set the configuration flags.

```
# rcctl set nxlog flags -c /tmp/sample-nxlog.conf
# rcctl get nxlog
nxlog_class=daemon
nxlog_flags=-c /tmp/sample-nxlog.conf
nxlog_rtable=0
nxlog_timeout=30
nxlog_user=root
# rcctl reload nxlog
```

8. Check the NXLog service status using [rcctl\(8\)](#).

```
# rcctl check nxlog
nxlog(ok)
```

11.2. Upgrading

To upgrade from a previous NXLog version (whether a licensed copy or trial), use the [pkg_add\(1\)](#) utility. This example shows an upgrade from version [3.0.1865](#) to [4.0.3550](#).

```
# pkg_add -U nxlog-4.0.3550-obsd6.tgz
nxlog-3.0.1865-obsd6\->4.0.3550-obsd6: ok
Read shared items: ok
```

To replace a trial installation of NXLog Enterprise Edition with a licensed copy of the same version, use [pkg_add](#) with the replace flag ([-r](#)).

```
# pkg_add -r nxlog-4.0.3550-obsd6.tgz
```

11.3. Uninstalling

To uninstall NXLog, follow these steps.

1. Use the [pkg_delete\(1\)](#) utility to remove the [nxlog](#) package.

```
# pkg_delete nxlog
nxlog-4.0.3550-obsd6: ok
Read shared items: ok
--- -nxlog-4.0.3550-obsd6 -----
You should also run /usr/sbin/userdel nxlog
```

2. Remove the [nxlog](#) user.

```
# /usr/sbin/userdel nxlog
```

3. Remove the [/opt/nxlog](#) directory.

```
# rm -rf /opt/nxlog
```

Chapter 12. Microsoft Windows

12.1. Installing

First, download the NXLog MSI file from the [NXLog website](#).

1. Login to your account, then click **My account** at the top of the page.
2. Under the **Downloads > NXLog Enterprise Edition files** tab, download **nxlog-4.0.3550.msi**.

There are several ways that NXLog can be installed on Windows.

- [Installing Interactively](#)
- [Installing With Msieexec](#)
- [Deploying via Group Policy](#)

12.1.1. Installing Interactively

1. Run the installer by double-clicking the MSI. Accept the license agreement, customize the installation directory if desired, and click **[Install]**. Click **[Finish]** when the installation completes; by default, the **README.txt** file will be opened in Notepad.
2. Find the configuration file (**nxlog.conf**) by navigating to the chosen installation directory and then the **conf** subdirectory. By default, the configuration file will be found at **C:\Program Files (x86)\nxlog\conf\nxlog.conf** on a 64-bit system or **C:\Program Files\nxlog\conf\nxlog.conf** on a 32-bit system.
3. Open **nxlog.conf** with Notepad. Update the **ROOT** path if you chose a custom installation directory.

```
# Default on 64-bit Windows
define ROOT C:\Program Files (x86)\nxlog
```

4. Configure NXLog by editing **nxlog.conf**. General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on Windows, see the [Microsoft Windows](#) summary.
5. Verify the configuration file syntax.

```
> "C:\Program Files (x86)\nxlog\nxlog.exe" -v
2017-03-17 08:05:06 INFO configuration OK
```

6. Start NXLog by opening the Service Manager, finding the **nxlog** service in the list, and starting it. To run the **nxlog.exe** executable in the foreground, rather than as a service, execute it with the **-f** command line argument.
7. Open the NXLog log file (default **C:\Program Files (x86)\nxlog\data\nxlog.log** on 64-bit Windows) with Notepad and check for errors.

NOTE

Some text editors (such as Wordpad) use exclusive locking and will refuse to open the log file while NXLog is running.

12.1.2. Installing With Msieexec

Msieexec can be used for performing an unattended install of NXLog. This command does not prompt the user at all, but must be run as administrator.

```
> msieexec /i nxlog-4.0.3550.msi /q
```

To allow Windows to prompt for administrator privileges, but otherwise install unattended, use `/qb` instead.

```
> msieexec /i nxlog-4.0.3550.msi /qb
```

To specify a custom installation directory, use the `INSTALLDIR` property. Remember to set the correct ROOT path in `nxlog.conf`.

```
> msieexec /i nxlog-4.0.3550.msi /q INSTALLDIR="C:\nxlog"
```

12.1.3. Deploying via Group Policy

For large deployments, it may be convenient to use Group Policy to manage the NXLog installation.

NOTE

These steps were tested with a Windows Server 2016 domain controller and a Windows 7 client. There are multiple ways to configure NXLog deployment with Group Policy, and the required steps for your network may vary from those listed below.

1. Log on to the server as an administrator.
2. Set up an Active Directory group for computers requiring an NXLog installation. NXLog will be automatically installed and configured on each computer in this group.
 - a. Open the **Active Directory Users and Groups** console (`dsa.msc`).
 - b. Under the domain, right-click on **Computers** and click **New > Group**.
 - c. Provide a name for the group (for example, `nxlog`). Use the **Security** group type and **Global** context (or the context suitable for your case).
 - d. Add computers to the group by selecting one or more, clicking **Actions > Add to a group...**, and entering the group name (`nxlog`).
3. Create a network share for distributing the NXLog files.
 - a. Create a folder in the desired location (for example, `C:\nxlog-dist`).
 - b. Set up the folder as a share: right-click, select **Properties**, open the **Sharing** tab, and click [**Share...**].
 - c. Add the group (`nxlog`) and click [**Share**]. Take note of the share name provided by the wizard, it will be needed later (for example, `\WINSERV1\nxlog-dist`).
 - d. Copy the required files to the shared folder. If using NXLog Manager, this will include at least three files: `nxlog-4.0.3550.msi`, `log4ensics.conf`, and CA certificate `agent-ca.pem`. If not using NXLog Manager, use a custom `nxlog.conf` instead of `log4ensics.conf`, omit the CA certificate, and include any other files required by the configuration.

NOTE

To deploy on both 32-bit and 64-bit systems, you will need a second `nxlog.conf` with ROOT configured as `C:\Program Files\nxlog`. See [Add the required files](#) below.

4. Create a Group Policy Object (GPO) for the NXLog deployment.
 - a. Open the **Group Policy Management** console (`gpmc.msc`).
 - b. In the console tree, under **Domains**, right-click on your domain and click **Create a GPO in this domain, and Link it here...**; this will create a GPO under the **Group Policy Objects** folder and link it to the domain.
 - c. Name the GPO (for example, `nxlog`) and click [**OK**].
 - d. Select the newly created GPO in the tree.
 - e. In the **Security Filtering** list, add the Active Directory group created in step 2 (`nxlog`). Remove anything else.

- f. Right-click on the GPO and click **Edit**. The **Group Policy Management Editor** console will be opened for editing the GPO.
5. Add the NXLog MSI to the GPO.

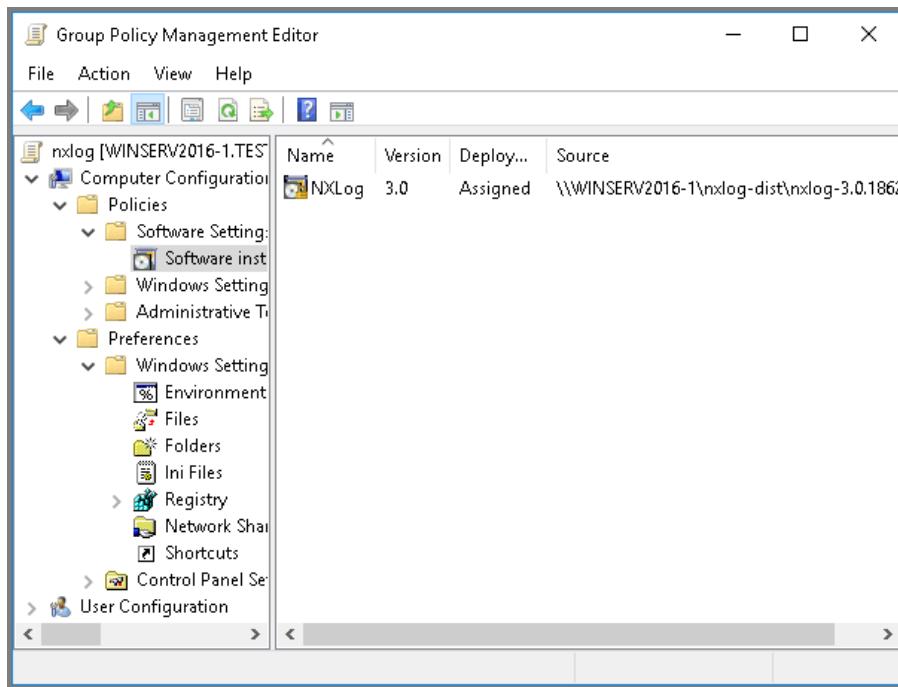


Figure 1. Configured NXLog GPO

- a. Under **Computer Configuration > Policies > Software Settings**, right-click **Software installation**. Click **New > Package...** to create a deployment package for NXLog.
- b. Browse to the network share and open the `nxlog-4.0.3550.msi` package. It is important to use the Uniform Naming Convention (UNC) path (for example, `\WINSERV1\ nxlog-dist`) so the file will be accessible by the remote computers.
- c. Select the **Assigned** deployment method.
6. Add the required files to the GPO by following these steps for each file.

NOTE

The following steps are for deployment on 64-bit systems only. To deploy NXLog on both 32-bit and 64-bit systems with a single GPO, use two entries for each file. For each entry, enable **Item-level targeting** (under the **Common** tab in the file's **Properties** dialog) to create the file with the correct path (`C:\Program Files (x86)\ nxlog` or `C:\Program Files\ nxlog`) based on whether the `%Processor_Architecture%` environment variable matches `AMD64` or `x86` respectively.

Also, if deploying `nxlog.conf` (not using NXLog Manager), there must be separate `nxlog32.conf` and `nxlog64.conf` source files, each containing the correct ROOT path.

- a. Under **Computer Configuration > Preferences > Windows Settings**, right-click on **Files**. Click **New > File**.
- b. Select the **Replace** action in the drop-down.
- c. Choose the source file on the network share (for example, `\WINSERV1\ nxlog-dist\ log4ensics.conf` or `\WINSERV1\ nxlog-dist\ agent-ca.pem`).
- d. Type in the destination path for the file (for example, `C:\Program Files (x86)\ nxlog\conf\log4ensics.conf` or `C:\Program Files (x86)\ nxlog\cert\agent-ca.pem`).
- e. Check **Apply once and do not reapply** under the **Common** tab for files that should only be deployed once. This is especially important for `log4ensics.conf` because NXLog Manager will write configuration

changes to that file.

f. Click **[OK]** to create the File in the GPO.

7. After the Group Policy is updated on the clients and NXLog is installed, one more reboot will be required before the NXLog service starts automatically.

For more information about Group Policy, see the following TechNet and MSDN articles:

- [Group Policy for Beginners](#),
- [Group Policy Planning and Deployment Guide](#),
- [Step-by-Step Guide to Understanding the Group Policy Feature Set](#), and
- [Step-by-Step Guide to Software Installation and Maintenance](#).

12.2. Upgrading

To upgrade the NXLog installation to the latest release, or to replace a trial installation of NXLog Enterprise Edition with a licensed copy, follow the steps below.

1. Run the new MSI installer as described in the [Installing](#) section (interactively, with Msiexec, or via Group Policy). The installer should detect that the previous version is installed and do an upgrade to the same installation directory.

NOTE

If you are upgrading from v3.x, you will need to manually uninstall the previous version first (see [Uninstalling](#)). This is necessary to transition from a per-user to a per-machine installation. You can skip this check by passing the **SKIP_PERUSER_CHECK** property (such as `msiexec /i nxlog-4.0.3550.msi /q SKIP_PERUSER_CHECK=1`). Note that this may leave behind another installation and its associated Add/Remove Programs entry.

NOTE

If the **Services** console (`services.msc`) is running, the installer may request the computer to be rebooted or give a permission denied error. Please ensure that the **Services** console is not running before attempting an upgrade.

2. Start the upgraded NXLog service with the **Services** console (`services.msc`) or by rebooting the system. Check the log file (default `C:\Program Files (x86)\nxlog\data\nxlog.log` on 64-bit Windows) and verify that logging is working as expected.

For Group Policy deployments, follow these steps:

1. Download the new MSI package as described in the [Installing](#) introduction.
2. Place the new MSI in the distribution share (see [Create a network share](#)).
3. Add this MSI as a new package to the NXLog GPO (follow the steps under [Add the NXLog MSI](#)).
4. Right-click on the new package and click **Properties**. Open the **Upgrades** tab, click **[Add...]**, select the previous version from the list, and click **[OK]**.

NOTE

If you want to downgrade to a previous version of NXLog, you will need to manually uninstall the current version first. See [Uninstalling](#).

12.3. Uninstalling

NXLog can be uninstalled from the Control Panel or with Msiexec and the original NXLog MSI.

WARNING

NXLog v3.x installers will remove `log4ensics.conf` and `nxlog.conf` on uninstallation, even if they have been modified. If you wish to keep these files, save them elsewhere before uninstalling NXLog v3.x.

Msiexec can be used to uninstall NXLog.

```
> msiexec /x nxlog-4.0.3550.msi /qb
```

NOTE

This procedure will not remove any configuration files, additional files created to set up NXLog, or files that were created as a result of NXLog's logging operations (except for v3.x installers as noted above). You may wish to remove the installation directory (default `C:\Program Files (x86)\nxlog` on 64-bit Windows) after completing the installation.

For Group Policy deployments, follow these steps:

1. Open the Group Policy Object (GPO) originally created for installation (see [Create a Group Policy Object](#)).
2. For each NXLog version that has been deployed, right-click the package and either:
 - click **All Tasks > Remove...**, and choose the **Immediately uninstall** removal method; or
 - click **Properties**, open the **Deployment** tab, and check **Uninstall this application when it falls out of the scope of management**.

NOTE

In this case, NXLog will be uninstalled when the GPO is no longer applied to the computer. An additional action will be required, such as removing the selected computer(s) from the `nxlog` group created in [Set up an Active Directory group](#).

12.4. Configure using a custom MSI

NXLog can be configured using a custom built MSI file. The MSI file will install the CA certificate and customized configuration files of your choosing which can be deployed alongside the NXLog MSI. Deployment of the configuration MSI can be done with the same methods as with the NXLog MSI, namely [interactively](#), [using msiexec](#) or via [group policy](#).

NOTE

Deployment via Group Policy already provides a way to deploy the configuration files. For this reason you might prefer to configure NXLog [via GPO](#) instead of creating a custom MSI described in this section.

The Windows Installer XML Toolset (Wix) is required to build the custom MSI. Wix is free software available for download from <http://wixtoolset.org>. A zip file containing all the assets, XML and scripts is also needed, that can be downloaded from NXLog web site.

Download and install Wix. Make a note where the binary folder of Wix is located (containing the `candle.exe` and `light.exe` executables, typically `C:\Program Files (x86)\WiX Toolset v3.11\bin`). Uncompress the provided zip file in a folder of your choosing and make sure the path to the binary folder is correct in the `pkgmsi.bat` script by editing the `WIX_BUILD_LOCATION` variable. Add the custom `agent-ca.pem` and `log4ensics.conf` files in the folder. The files to be deployed can be customized by editing `nxlog-conf.wxs`.

Finally, execute the `pkgmsi.bat` script. The script will request the desired architecture (x86 for 32-bit NXLog or x64 for 64-bit NXLog) and once entered, it will proceed to build the MSI. Depending on the architecture selected the result will either be an `nxlog-conf_x86.msi` or a `nxlog-conf_x64.msi`. You can now deploy the custom configuration MSI alongside with NXLog.

Chapter 13. Oracle Solaris

13.1. Installing

1. Download the appropriate NXLog install archive from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > My downloads** tab, choose the correct archive for your system.

Table 47. Available Solaris Files

Platform	Archive
Solaris 10/11 x86 archive	nxlog-4.0.3550_solaris-x86.pkg.gz
Solaris 10/11 SPARC archive	nxlog-4.0.3550_solaris-sparc.pkg.gz

2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Log in to the target server and extract the contents of the archive.

```
$ gunzip nxlog-4.0.3550_solaris-sparc.pkg.gz
```

4. Install the NXLog package.
 - For interactive installation, issue the following command and answer **y** (yes) to the questions.

```
$ sudo pkgadd -d nxlog-4.0.3550.pkg NXnxlog
```

- For a quiet install, use an administration file. Place the file (**nxlog-adm** in this example) in the **/var/sadm/install/admin/** directory.

```
$ sudo pkgadd -n -a nxlog-adm -d nxlog-4.0.3550.pkg NXnxlog
```

nxlog-adm

```
mail=
instance=overwrite
partial=nocheck
runlevel=nocheck
idepend=nocheck
rdepend=nocheck
space=quit
setuid=nocheck
conflict=nocheck
install
action=nocheck
basedir=/opt/nxlog
networktimeout=60
networkretries=3
authentication=quit
keystore=/var/sadm/security
proxy=
```

5. Configure NXLog by editing **/opt/nxlog/etc/nxlog.conf**. General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on Solaris, see the [Oracle Solaris](#) summary.
6. Verify the configuration file syntax.

```
$ sudo /opt/nxlog/bin/nxlog -v
2017-03-17 08:05:06 INFO configuration OK
```

7. Check that the NXLog service is running with the **svcs** command.

```
$ svcs nxlog  
online    12:40:37 svc:system/nxlog:default
```

8. Manage the NXLog service with **svcadm** (restart the service to load the edited configuration file).

```
$ sudo svcadm restart nxlog  
$ sudo svcadm enable nxlog  
$ sudo svcadm disable nxlog
```

13.2. Upgrading

To update an NXLog installation to the latest release, use **pkgadd** as in the [installation instructions](#) above.

- For interactive installation, issue the following command and answer **y** (yes) to the questions.

```
$ sudo pkgadd -d nxlog-4.0.3550.pkg NXnxlog
```

- For a quiet install, use an administration file.

```
$ sudo pkgadd -n -a nxlog-adm -d nxlog-4.0.3550.pkg NXnxlog
```

To replace a trial installation of NXLog Enterprise Edition with a licensed copy of the same version, follow the same [installation instructions](#) (use **instance=overwrite** as shown).

13.3. Uninstalling

To uninstall NXLog, use **pkgrm**. To remove the package files from the client's file system, include the **-A** option.

```
$ sudo pkgrm NXnxlog
```

NOTE

This procedure may not remove all files that were created in order to configure NXLog, or that were created as a result of NXLog's logging operations. To find these files, consult the configuration files that were used with NXLog and check the installation directory ([/opt/nxlog](#)).

Chapter 14. IBM AIX

14.1. Installing

1. Download the appropriate NXLog installer package from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the **nxlog-4.0.3550-1.aix6.1.ppc.rpm** package.
2. Use SFTP or a similar secure method to transfer the archive to the target server.
3. Install the required NXLog package.

```
$ sudo rpm -ivh nxlog-4.0.3550-1.aix6.1.ppc.rpm
```

4. Configure NXLog by editing **/opt/nxlog/etc/nxlog.conf**. See the [IBM AIX](#) section for AIX specific configuration. General information about configuring NXLog can be found in [Configuration](#).
5. Verify the configuration file syntax.

```
$ sudo /opt/nxlog/bin/nxlog -v  
2017-03-17 08:05:06 INFO configuration OK
```

6. Start the service using the **init** script in **/opt/nxlog/etc**:

```
$ sudo ./init start
```

14.2. Upgrading

To update an NXLog installation to the latest release, use **rpm** as in the [installation instructions](#) above.

```
$ sudo rpm -Uvh nxlog-4.0.3550-1.aix6.1.ppc.rpm
```

NOTE

The rpm package manager creates a backup of an existing **nxlog.conf** file as **nxlog.conf.rpmsave** under the **/opt/nxlog/etc/** directory.

14.3. Uninstalling

To uninstall NXLog use **rpm** with the **-e** option.

```
$ sudo rpm -e nxlog
```

NOTE

This procedure may not remove all files that were created in order to configure NXLog, or that were created as a result of NXLog's logging operations. To find these files, consult the configuration files that were used with NXLog and check the installation directory (**/opt/nxlog**).

Chapter 15. Apple macOS

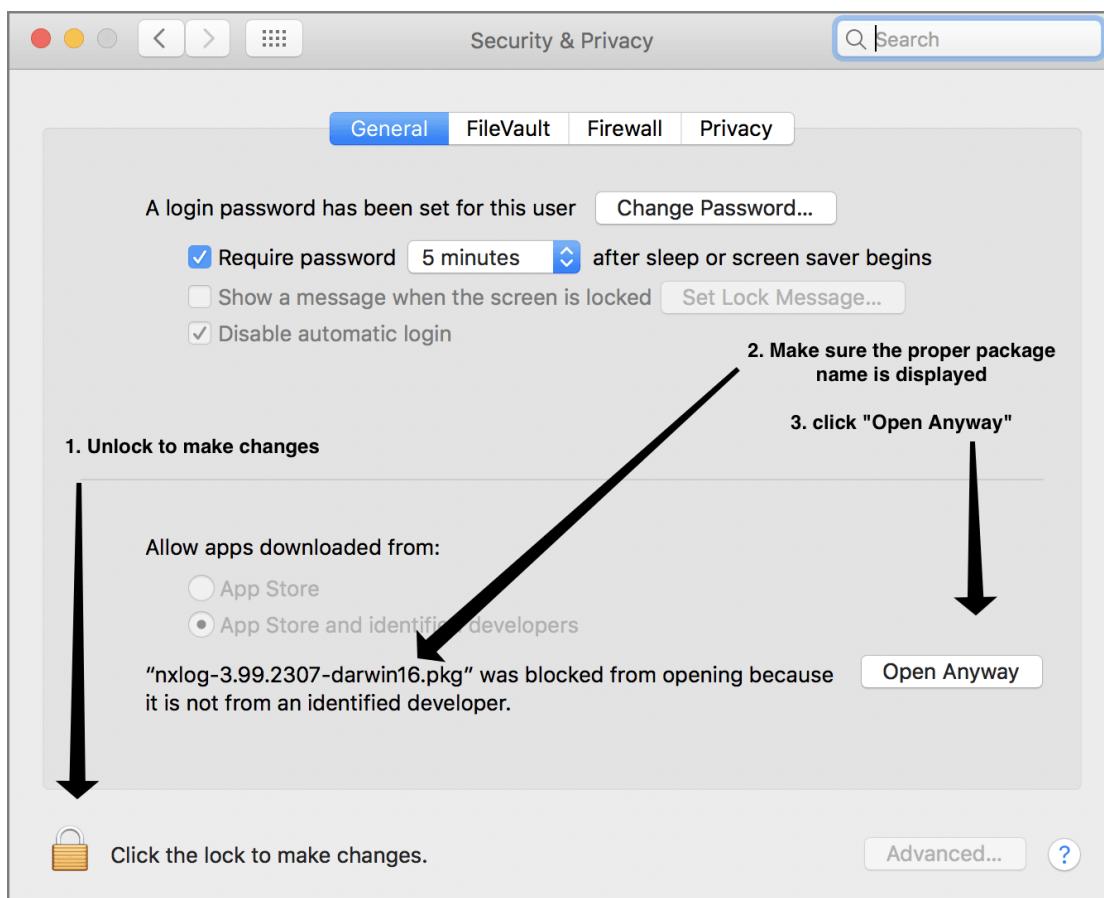
15.1. Installing

To install NXLog in macOS, follow these steps. You will need administrator privileges to complete the installation process.

1. Download the appropriate NXLog install package from the [NXLog website](#).
 - a. Login to your account, then click **My account** at the top of the page.
 - b. Under the **Downloads > NXLog Enterprise Edition files** tab, choose the **nxlog-4.0.3550-macos-x86.pkg** package.
2. Install the NXLog package. You can do the installation interactively or with the command line installer.
 - To install interactively, double-click the NXLog package.

If you see a message like the following, go to **System Preferences > Security & Privacy** and click [**Open Anyway**]. Then follow the installer's instructions.

"nxlog-4.0.3550-macos-x86.pkg" can't be opened because it is from an unidentified developer.



- To install the package using the command line installer, run the following command.

```
$ sudo installer -pkg nxlog-4.0.3550-macos-x86.pkg -target /
Password:
installer: Package name is nxlog-4.0.3550-macos-x86
installer: Upgrading at base path /
installer: The upgrade was successful.
```

Upon installation, all NXLog files are placed under `/opt/nxlog`. The `launchd(8)` script is installed in `/Library/LaunchDaemons/com.nxlog.plist` and has the `KeepAlive` flag set to `true` (launchd will automatically restart NXLog). NXLog log files are managed by launchd and can be found in `/var/log/`.

3. Configure NXLog by editing `/opt/nxlog/etc/nxlog.conf`. General information about configuring NXLog can be found in [Configuration](#). For more details about configuring NXLog to collect logs on macOS, see the [Apple macOS](#) summary.
4. Verify the configuration file syntax.

```
$ sudo /opt/nxlog/bin/nxlog -v  
2017-03-17 08:05:06 INFO configuration OK
```

5. To apply your changes, stop NXLog with the following command. The launchd manager will restart the daemon and the new configuration will be loaded.

```
$ sudo launchctl stop com.nxlog
```

6. To permanently stop NXLog, the service must be unloaded.

```
$ sudo launchctl unload /Library/LaunchDaemons/com.nxlog.plist
```

15.2. Upgrading

To upgrade NXLog, follow the [installation instructions](#).

The installation script will not modify the configuration files, and the script will restart NXLog after the installation is completed.

15.3. Uninstalling

To properly uninstall NXLog, follow these steps.

1. Unload the daemon.

```
$ sudo launchctl unload /Library/LaunchDaemons/com.nxlog.plist
```

2. Delete the `nxlog` user and group that were created during installation.

```
$ sudo dscl . -delete "/Groups/nxlog"  
$ sudo dscl . -delete "/Users/nxlog"
```

3. Remove NXLog files.

WARNING

This will remove any custom configuration, certificates, or other files in the listed directories. Save these files to another location first if you do not wish to discard them.

```
$ sudo rm -rf /opt/nxlog /Library/LaunchDaemons/com.nxlog.plist /var/log/nxlog.std* && sudo  
pkgutil --forget com.nxlog.agent
```

Chapter 16. Hardening NXLog

16.1. Running Under a Non-Root User on Linux

NXLog can be configured to improve security by running under a specified user and group rather than retaining root privileges. The [User](#) and [Group](#) global directives specify the user and group for the NXLog process to run under. By default on Linux installations, NXLog is configured to run under the `nxlog` user and `nxlog` group as shown below.

Running as nxlog:nxlog

```
1 User nxlog  
2 Group nxlog
```

Some operations require privileges that are normally not available to the `nxlog` user. In this case, the simplest solution is to configure NXLog to retain full root privileges by removing the [User](#) and [Group](#) directives from the configuration. This is not recommended, however; it is more secure to provide only the required privileges and to avoid running NXLog as root. See the following sections for more information.

16.1.1. Reading From /var/log

By default, the `nxlog` user will not have access to files in `/var/log`. If your Linux distribution uses a group other than root for the log files, you can use that group with the [Group](#) directive. Otherwise, reconfigure your system logger (Rsyslog for example) to create files with the necessary ownership. See [Reading Rsyslog Log Files](#) for more information.

16.1.2. UDP Spoofing and Binding to Ports Below 1024

NXLog requires special privileges if configured to perform UDP source address spoofing (with [om_udpspoof](#)) or to bind to a port below 1024 (for example to accept incoming Syslog messages on port 514). Consider the following solutions.

Use built-in capability support

NXLog will automatically set the corresponding Linux capability before dropping root privileges.

Set the capability manually

For binding to ports below 1024, use the `CAP_NET_BIND_SERVICE` capability. For the UDP source address spoofing, use the `CAP_NET_RAW` capability.

Example 7. Setting Linux Capabilities

This command sets the `CAP_NET_BIND_SERVICE` capability for the NXLog executable.

```
# setcap cap_net_bind_service+ep /opt/nxlog/bin/nxlog
```

This command sets both the `CAP_NET_BIND_SERVICE` and the `CAP_NET_RAW` capabilities.

```
# setcap cap_net_bind_service,cap_net_raw=+ep /opt/nxlog/bin/nxlog
```

Verify with this command, or by adding the `-v` (verify) flag to the `setcap` command.

```
# getcap /opt/nxlog/bin/nxlog
```

16.1.3. Reading the Kernel Log

NXLog requires special privileges to read from the Linux kernel log with the [im_kernel](#) module. Consider the

following solutions.

Use built-in capability support

NXLog will automatically set the Linux CAP_SYS_ADMIN capability before dropping root privileges.

Set the capability manually

Use the CAP_SYS_ADMIN capability or the CAP_SYSLOG capability (since Linux 2.6.37). See [Setting Linux Capabilities](#).

16.2. Configuring SELinux

To further harden NXLog, SELinux can optionally be used. SELinux improves security by providing mandatory access controls on Linux through *policies*. This guide shows you how to configure an SELinux policy for NXLog. The resulting policy will provide the permissions necessary for your NXLog deployment to operate as configured, with SELinux enabled on the host.

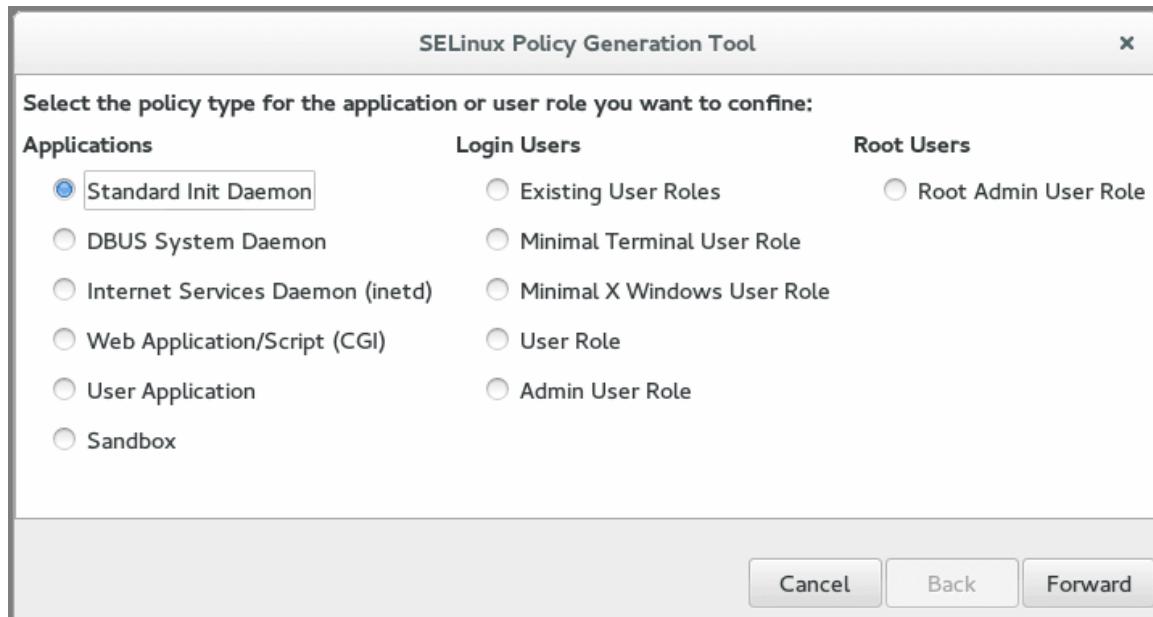
1. First, [use the SELinux Policy Generation Tool](#) to create a base policy file.
2. Second, you can [optionally use the audit2allow tool](#) to tailor the policy for your specific deployment environment.

16.2.1. Base Policy

1. Install the SELinux Policy Generation Tool package.

```
$ sudo yum install policycoreutils-gui
```

2. Start the SELinux Policy Generation Tool from the system launcher.
3. In the first screen, select **Standard Init Daemon** for the policy type, then click [**Forward**].



4. On the second screen, enter the following details for the application and user role, then click [**Forward**].

Name

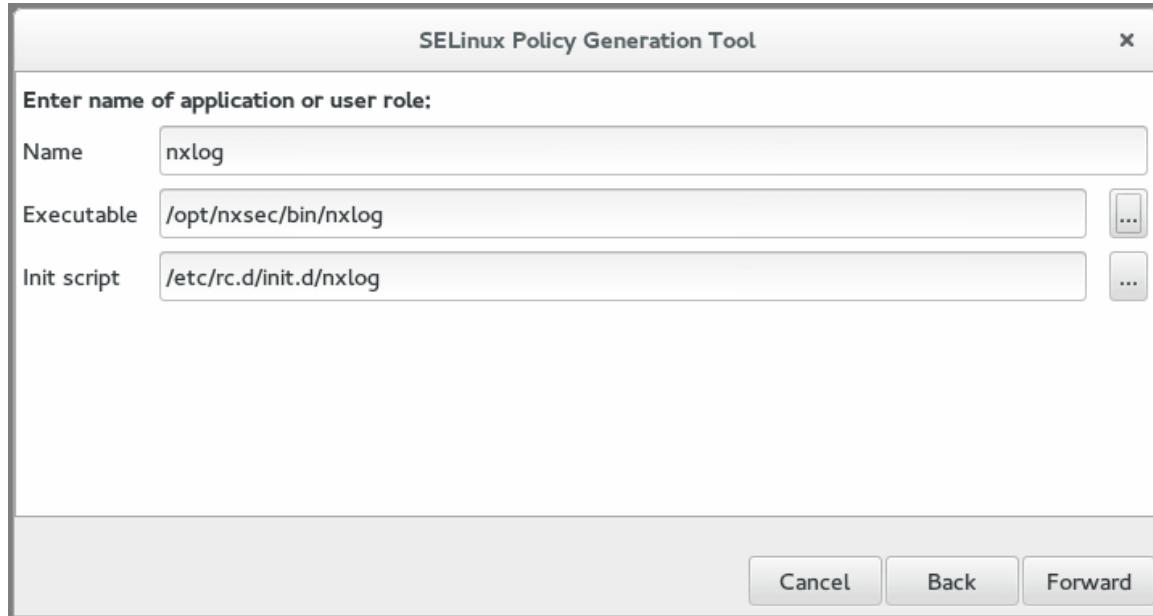
A custom name for the role (for example, **nxlog**)

Executable

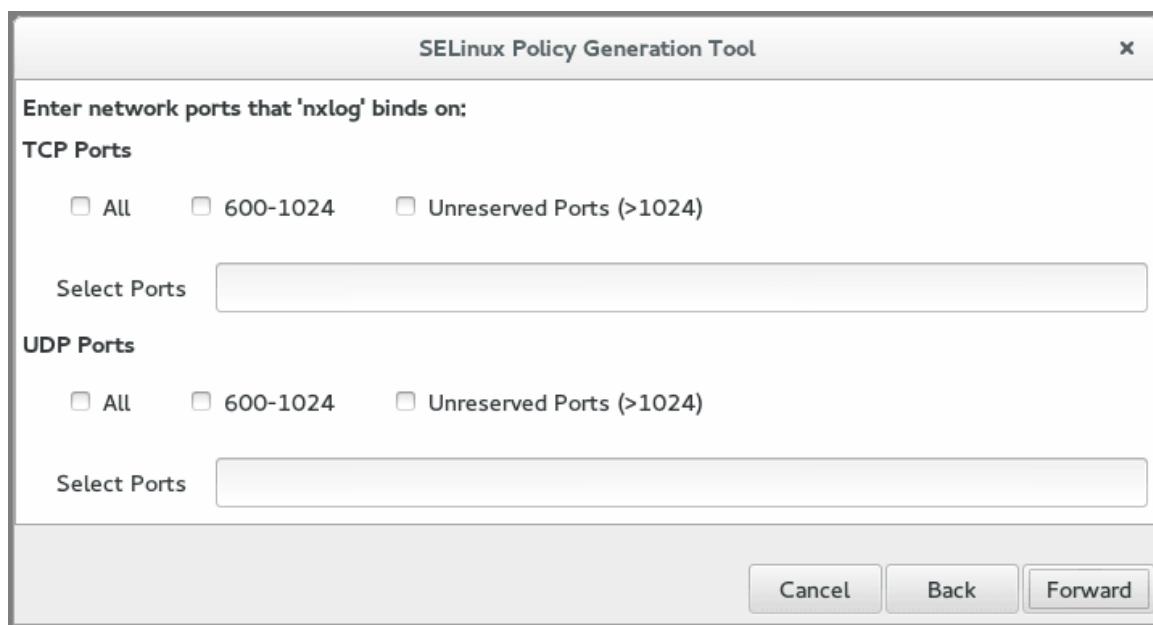
The path to the NXLog executable (for example, **/opt/nxlog/bin/nxlog**)

Init script

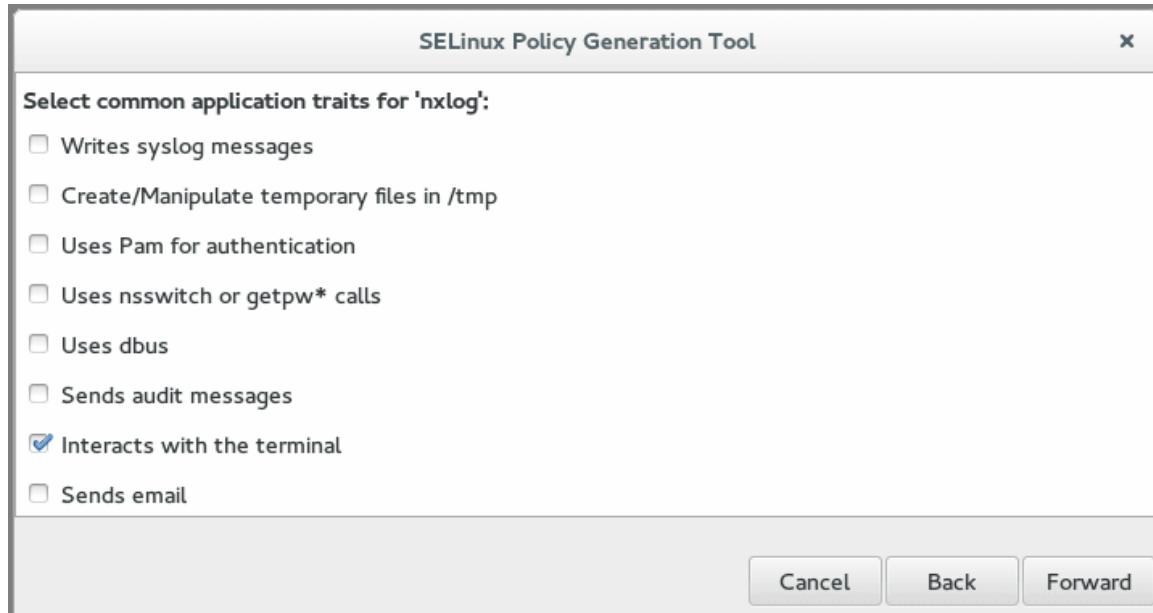
The path of the NXLog system init script (for example, `/etc/rc.d/init.d/nxlog`)



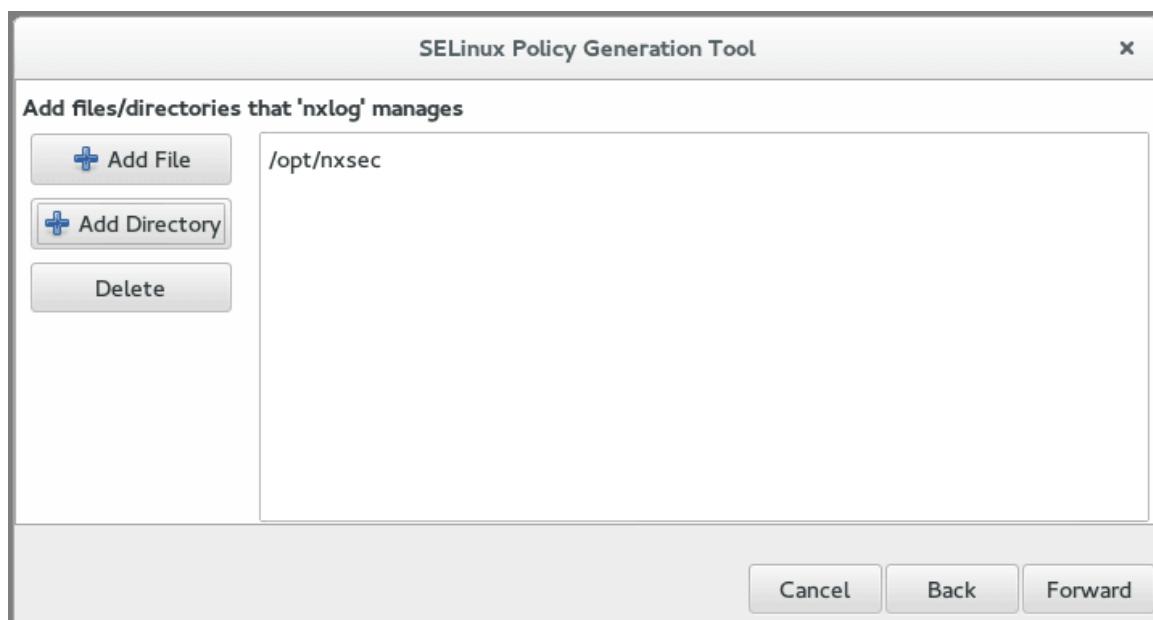
5. On the third screen, enter the TCP and UDP used by the NXLog deployment, then click [**Forward**]. If the ports are unknown or not yet determined, then leave these fields blank; they can be customized later.



6. On the fourth screen, select the appropriate application traits for NXLog, then click [**Forward**]. The default configuration requires only the **Interacts with the terminal** trait. For collecting Syslog messages or creating files in /tmp, include the appropriate traits.



- On the fifth screen, specify all the arbitrary files and directories that the NXLog installation should have access to, then click [**Forward**]. The default configuration requires only the NXLog system directory, `/opt/nxlog`. Include the paths of any custom log files that NXLog needs to access.



- Additional SELinux configuration values can be set on the sixth screen. None of these are required for NXLog. Click [**Forward**] to continue.
- The policy files are generated on the final screen. Click [**Save**] to write the policy to disk. Several files will be generated.

nxlog.te

Base policy information

nxlog.fc

File system information

nxlog.if

Interface information

nxlog.sh

A script for compiling and deploy the policy module, for use only on the target system

nxlog.spec

A specification file that can be used to generate an RPM package from the policy, useful for deploying the policy on another system later

16.2.2. Deploying and Customizing the Policy

In this section, the base policy generated in the [previous section](#) will be applied and then customized with appropriate rules for NXLog operation as configured. To accomplish this, SELinux will be set to permissive mode and then the *audit2allow* tool will be used to generate additional SELinux rules based on the resulting audit logs.

WARNING

When set to *permissive* mode, SELinux generates alerts rather than actively blocking actions as it does in *enforcing* mode. Because this reduces system security, it is recommended that this be done in a test environment.

1. Transfer the files containing your SELinux base policy to the target system. All the files should be in the same directory.
2. Apply the SELinux base policy by executing the policy script. This script will compile the policy module, set the appropriate security flags on the directories specified, and install the policy.

```
$ sudo ./nxlog.sh
```

3. Verify that the new policy is installed.

```
$ sudo semodule -l
```

4. Make sure that NXLog is correctly configured with all required functionality.
5. Set SELinux to permissive mode. All events which would have been prevented by SELinux will now be permitted and logged to /var/log/audit/audit.log (including events not related to NXLog).

```
$ sudo setenforce 0
```

6. Restart NXLog.

Example 8. Audit Logs

If NXLog has been configured to listen on TCP port 1514 with the *im_tcp* module, but that port is not specified in the current SELinux policy, then an audit log will be generated when the NXLog process initializes and binds to that port.

audit.log

```
type=AVC msg=audit(1404834389.548:444): avc: denied { accept } for pid=2224 comm="nxlog" laddr=127.0.0.1 lport=1514 scontext=unconfined_u:system_r:nxlog_t:s0 tcontext=unconfined_u:system_r:nxlog_t:s0 tclass=tcp_socket
```

Additional log messages will be generated for any other file or network action not permitted by the SELinux policy. These actions would all be denied by SELinux when set to enforcing mode.

7. The policy can be extended with additional rules with the help of the *audit2allow* tool, which will create SELinux rule statements for the NXLog audit log entries.

```
$ grep nxlog /var/log/audit/audit.log | audit2allow -R
require {
    type nxlog_rw_t;
    type nxlog_t;
    class capability { kill dac_override };
    class tcp_socket { accept listen };
    class process { signal getcap setcap };
    class file execute;
}

===== nxlog_t =====

#!!!! This avc is allowed in the current policy
allow nxlog_t nxlog_rw_t:file execute;

#!!!! This avc is allowed in the current policy
allow nxlog_t self:capability { kill dac_override };

#!!!! This avc is allowed in the current policy
allow nxlog_t self:process { signal getcap setcap };

#!!!! This avc is allowed in the current policy
allow nxlog_t self:tcp_socket { accept listen };
corenet_tcp_bind_generic_port(nxlog_t)
```

8. Copy everything in the `nxlog_t` section of the `audit2allow` output to the `nxlog.te` file.
9. Run the policy script again.

```
$ sudo ./nxlog.sh
```

10. Restart NXLog.
11. Check the audit log again for alerts related to NXLog. You should not see any such alerts. If you do see any alerts related to NXLog, then repeat the `audit2allow` steps to isolate the alerts into new rules.
12. Set the SELinux policy to enforcing mode. This can be set permanently in `/etc/selinux/config`.

```
$ sudo setenforce 1
```

13. Reboot the system.

16.3. Running Under a Custom Account on Windows

On Windows, the NXLog installer sets up the NXLog service to run under the local `system` account. This procedure describes how to configure a system service for NXLog that runs under a dedicated `svc-nxlog` user account. This approach can improve security by limiting the privileges that NXLog requires to run.

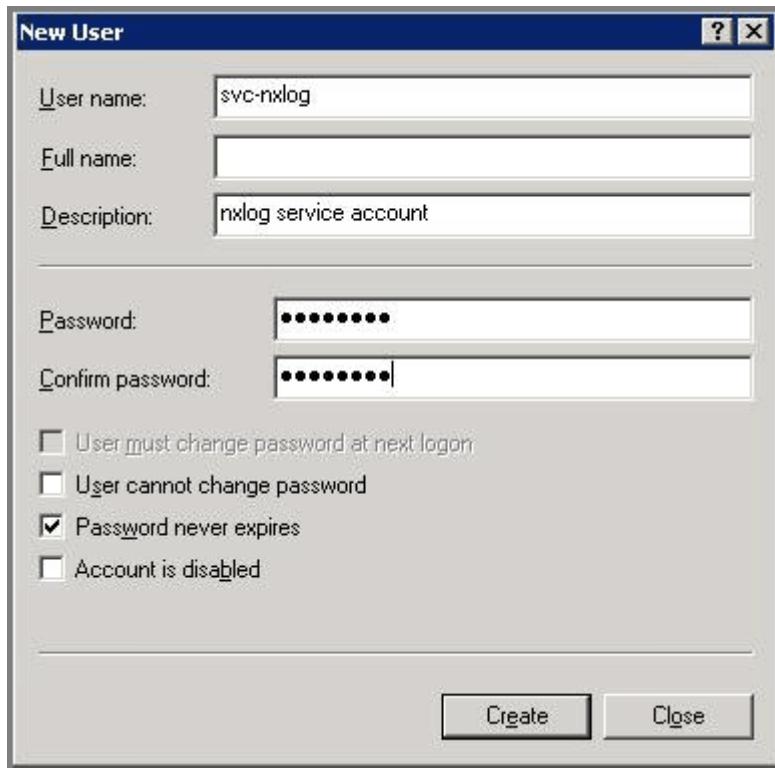
NOTE

In enterprise environments managed by Group Policy, the dedicated user account and its permissions must be managed by the domain administrator.

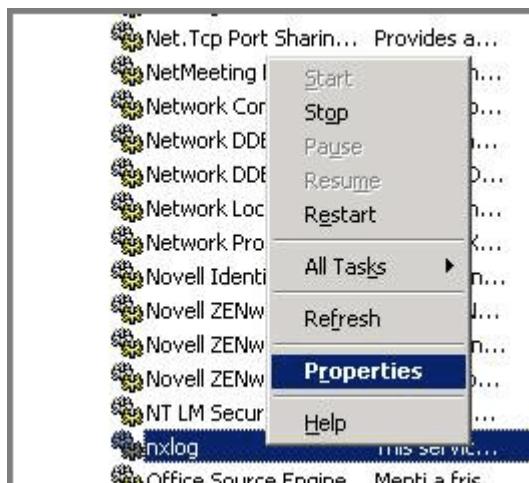
1. Create a new user account. Open the **Computer Management** console (`compmgmt.msc`), right-click on **Local Users and Groups > Users**, and select **New User...** from the context menu.



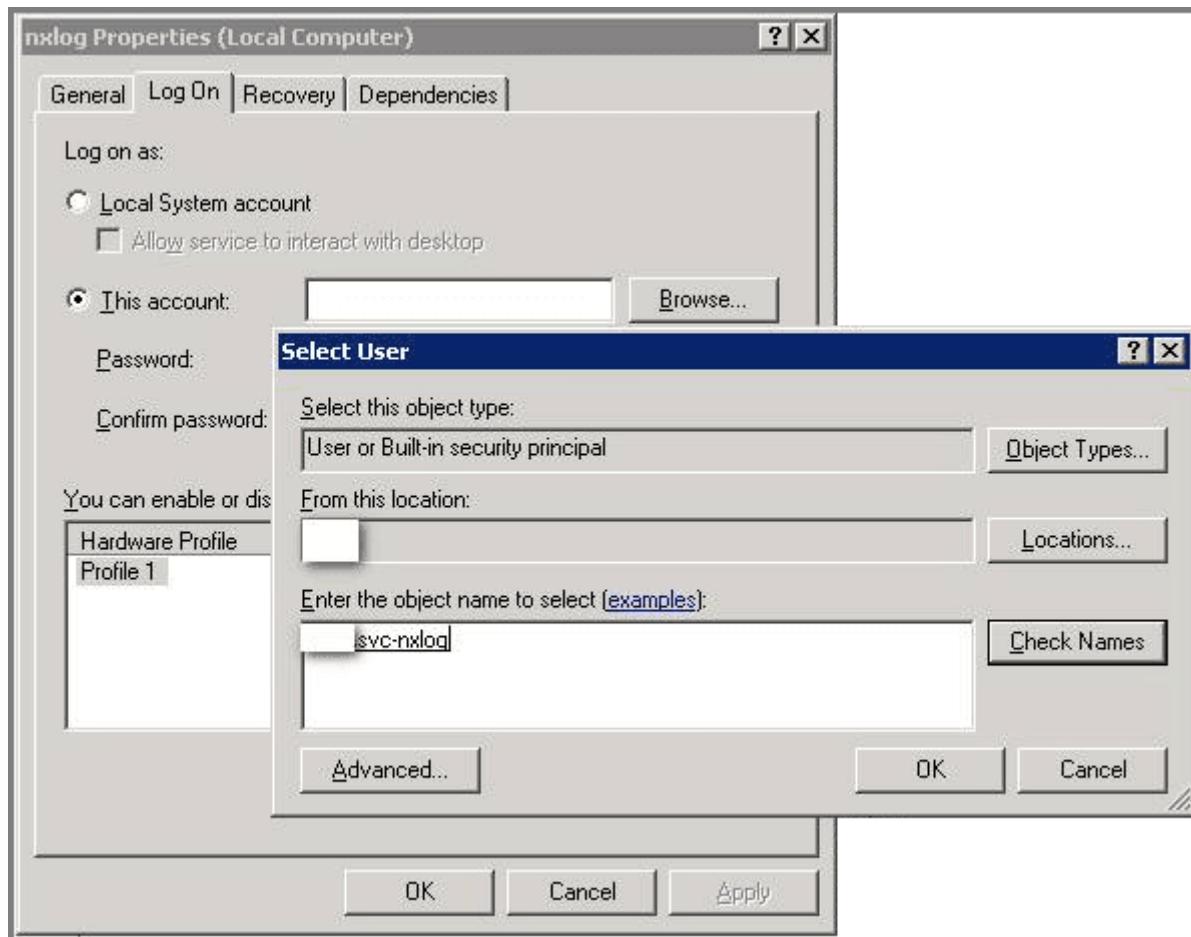
2. Enter the **svc-nxlog** user name, description, and password; enable the **Password never expires** check box; and click [**Create**].



3. Open the **Services** console (`services.msc`), right-click the **nxlog** service, and select **Properties**.



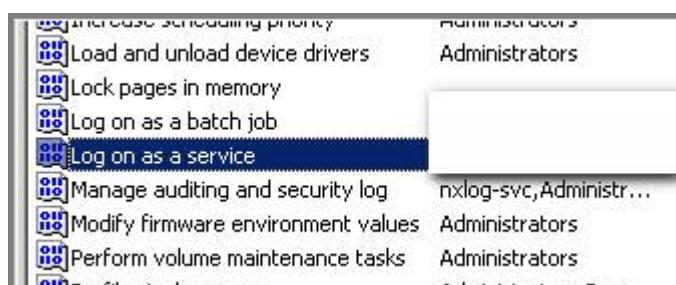
4. Under the **Log On** tab, select the **This Account** radio button, click [**Browse...**], select the **svc-nxlog** user account, and enter the password. Then click [**OK**]. Windows will warn you that the service must be restarted.



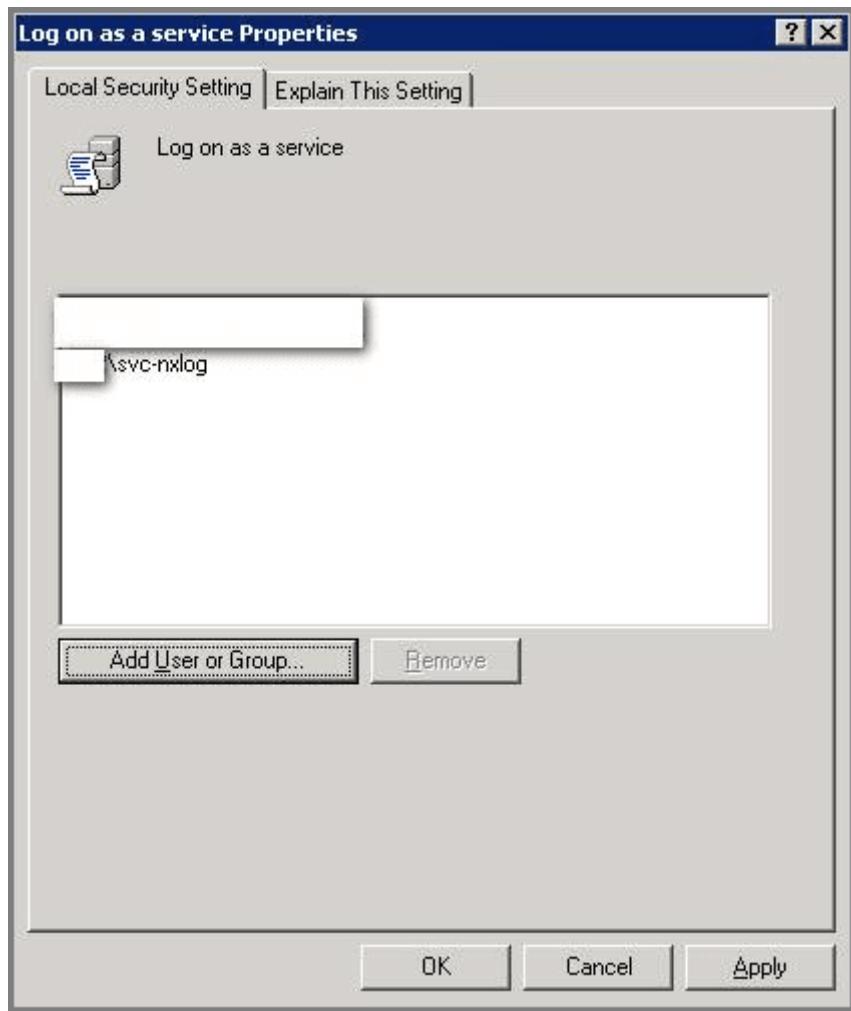
5. Open the **Local Security Settings** console (`secpol.msc`) and select **Local Policies > User Rights Assignment** in the left pane.



6. Right-click the **Log on as a service** policy and click **Properties**.



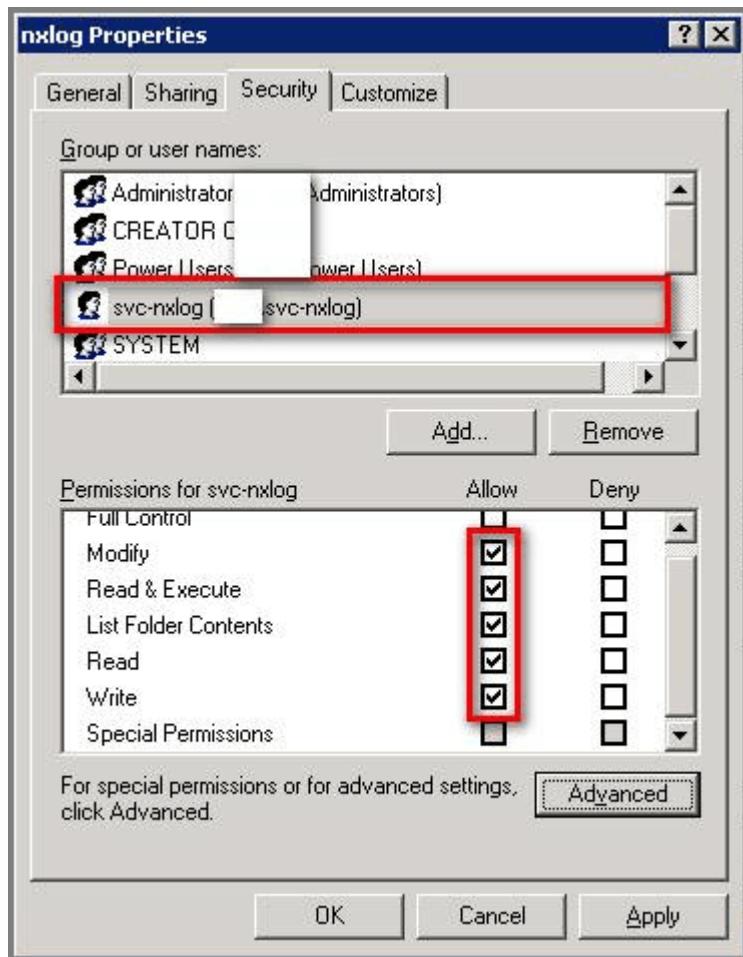
7. Click [**Add User or Group...**] and select the new user. The new user should appear in the list. Click [**OK**].



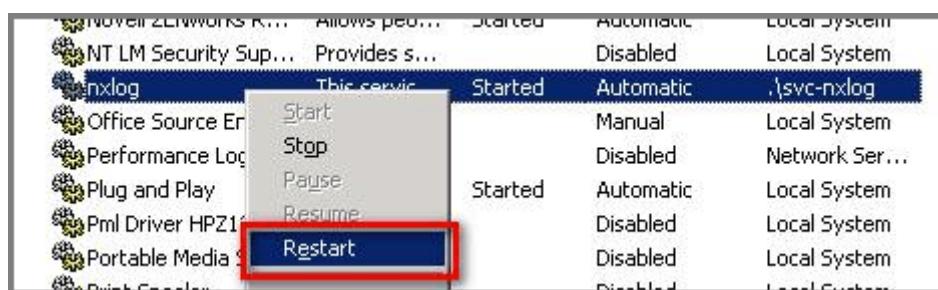
8. Add the new user to the the **Manage auditing and security log** policy also.

9. In Windows Explorer, browse to the NXLog installation directory (by default, `C:\Program Files (x86)\nxlog` on 64-bit systems), right-click, and select **Properties >]**. Under the **Security tab** > select the new user from the **Group or user names list**. Check **Allow for the following permissions >** and then click [**OK**].

- Modify
- Read & Execute
- List Folder Contents
- Read
- Write



10. In the **Services** console (`services.msc`), right-click the **nxlog** service and select **Restart**.



11. Check the NXLog log files for start-up errors. Successful startup should look like this:

`nxlog.log`

```
2016-11-16 16:53:10 INFO nxlog-4.0.3550 started<
2016-11-16 16:53:10 INFO connecting to 192.168.40.43<
2016-11-16 16:53:12 INFO successfully connected to 192.168.40.43:1514<
2016-11-16 16:53:12 INFO successfully connected to agent manager at 192.168.40.43:4041 in SSL
mode<
```

On some Windows systems, this procedure may result in the following access denied error when attempting to access the Windows EventLog:

NOTE WARNING ignoring source as it cannot be subscribed to (error code: 5)

In this case, `wEvtutil` can be used to set ACLs on the Windows EventLog. For more details, see the [Giving Non Administrators permission to read Event Logs Windows 2003 and Windows 2008](#) TechNet article.

Chapter 17. Relocating NXLog

While not officially supported, it is possible to relocate NXLog to a different directory than where it was installed originally. The procedure shown below assumes that NXLog was installed normally, using the system's package manager. While it is also possible to manually extract the files from the package and perform a manual installation in a custom directory, this is not covered here but the basic principals are the same. This procedure has been tested in GNU/Linux systems and should work in any system that supports run-time search paths.

WARNING

Both relocation and manual installation can result in a non-functional NXLog agent. Furthermore, subsequent update and removal using the system's package manager may not work correctly. Follow this procedure at your own risk. This is not recommended for inexperienced users.

Move the NXLog directory structure to the new location. Though not required, it is best to keep the original directory structure. Then proceed to the following sections.

NOTE

In the examples that follow, NXLog is being relocated from `/opt/nxlog` to `/opt/nxlog_new`.

17.1. System V Init File

For systems that manage services with System V, edit the NXLog init file. This file can normally be found at `/etc/init.d/nxlog`. Modify the init file so that the `$BASE` variable reflects the new directory. Update the `$pidfile`, `$nxlog`, and `$conf` variables to reference `$BASE`. Finally, reassign the `$nxlog` variable to include the configuration file. This must be done after any tests to the binary executable. The init file should look similar to the following.

```
/etc/init.d/nxlog
BASE=/opt/nxlog_new

pidfile=$BASE/var/run/nxlog/nxlog.pid
nxlog=$BASE/bin/nxlog
conf=$BASE/etc/nxlog.conf

test -f $nxlog || exit 0

nxlog="$nxlog -c $conf"
```

On systems that use a hybrid System V and Systemd, reload the init files by executing the following command.

```
# systemctl daemon-reload
```

17.2. Systemd Unit File

For systems using Systemd, the `/lib/systemd/system/nxlog.service` unit file must be edited and the paths updated.

nxlog.service

```
[Service]
Type=simple
User=root
Group=root
PIDFile=/opt/nxlog_new/var/run/nxlog/nxlog.pid
ExecStartPre=/opt/nxlog_new/bin/nxlog -v -c /opt/nxlog_new/etc/nxlog.conf
ExecStart=/opt/nxlog_new/bin/nxlog -f -c /opt/nxlog_new/etc/nxlog.conf
ExecStop=/opt/nxlog_new/bin/nxlog -s -c /opt/nxlog_new/etc/nxlog.conf
ExecReload=/opt/nxlog_new/bin/nxlog -r -c /opt/nxlog_new/etc/nxlog.conf
KillMode=process
```

Reload the modified unit files by executing the following command.

```
# systemctl daemon-reload
```

17.3. Edit Configuration File

The configuration file of NXLog itself must be modified to reflect the directory relocation, as well as any changes in the directory structure. For most cases, running `sed -i s,/opt/nxlog,/opt/nxlog_new,g /opt/nxlog_new/etc/nxlog.conf` will update the configuration file. Alternatively, edit the file manually as shown below.

nxlog.conf

```
define BASE /opt/nxlog_new
define CERTDIR %BASE%/var/lib/nxlog/cert
define CONFDIR %BASE%/var/lib/nxlog
define LOGDIR %BASE%/var/log/nxlog
define LOGFILE "%LOGDIR%/nxlog.log"

SpoolDir %BASE%/var/spool/nxlog

# default values:
PidFile %BASE%/var/run/nxlog/nxlog.pid
CacheDir %BASE%/var/spool/nxlog
ModuleDir %BASE%/lib/nxlog/modules
```

NOTE

Depending on the architecture and whether system supplied libraries are used, NXLog will store the modules under a different directory such as `%BASE%/libexec/nxlog/modules`.

17.4. Modify rpath

Depending on the NXLog package used, the run-time search path of the binaries must be changed. This is relevant for the generic versions of NXLog where the libraries are statically linked against the binaries. To check the shared libraries used by the binaries, issue the following command.

```
# ldd nxlog
    linux-vdso.so.1 => (0x00007ffc15d36000)
    libpcre.so.1 => /opt/nxlog/lib/libpcre.so.1 (0x00007ff7f311e000)
    libdl.so.2 => /lib64/libdl.so.2 (0x00007ff7f2f14000)
    libcap.so.2 => /lib64/libcap.so.2 (0x00007ff7f2d0f000)
    libapr-1.so.0 => /opt/nxlog/lib/libapr-1.so.0 (0x00007ff7f2ad9000)
    libert.so.1 => /lib64/libert.so.1 (0x00007ff7f28d0000)
    libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007ff7f2699000)
    libpthread.so.0 => /lib64/libpthread.so.0 (0x00007ff7f247d000)
    libc.so.6 => /lib64/libc.so.6 (0x00007ff7f20bb000)
    /lib64/ld-linux-x86-64.so.2 (0x00007ff7f336d000)
    libattr.so.1 => /lib64/libattr.so.1 (0x00007ff7f1eb6000)
    libfreebl3.so => /lib64/libfreebl3.so (0x00007ff7f1cb3000)
```

Notice that **libpcre** and **libapr** are pointing to the included libraries in **/opt/nxlog/lib/**. To change the run-time search path of the binaries, a tool such as **chrpath** or **patchelf** can be used.

```
# chrpath -r /opt/nxlog_new/lib nxlog
nxlog: RPATH=/opt/nxlog/lib:/home/builder/workspace/nxlog3-rpm-generic-
amd64/rpmbuild/BUILD/nxlog-deps/opt/nxlog/lib
nxlog: new RPATH: /opt/nxlog_new/lib
```

NXLog modules are also linked against statically included libraries. Therefore, if the run-time search path of the binaries required a change, then the rpath of the modules needs updated as well. To change the run-time search path of all the modules (or binaries) in a directory, use a command like this.

```
# chrpath -r /opt/nxlog_new/lib *
```

NXLog is now successfully relocated to a new directory.

Configuration

Chapter 18. Configuration Overview

NXLog uses [Apache style](#) configuration files. The configuration file is loaded from its default location, or it can be explicitly specified with the `-c` command line argument.

The configuration file is made up of blocks and directives. Blocks are similar to XML tags containing multiple directives. Directive names are case insensitive but arguments are case sensitive in some cases. A directive and its argument must be specified on the same line. Values spanning multiple lines must have the newline escaped with the backslash (\). A typical case for this is the `Exec` directive. Blank lines and lines starting with the hash mark (#) are ignored. Configuration directives referring to a file or a path can be quoted with double quotes ("") or single quotes (''). This applies to both global and module directives.

The configuration file can be logically divided into three parts: global parameters, module definitions and configurations, and routes (which link the modules together according to the data flow required).

Example 9. Configuration File Structure

Here are the three sections of a configuration file. This example contains two global directives, Input and Output blocks, and a Route block.

`nxlog.conf`

```
1 # Global section
2 User      nxlog
3 Group     nxlog
4
5 # Modules section
6 <Input in>
7   Module  im_null
8 </Input>
9
10 <Output out>
11   Module om_null
12 </Output>
13
14 # Route section
15 <Route r>
16   Path    in => out
17 </Route>
```

18.1. Global Directives

The global section contains directives that control the overall behavior of NXLog.

The [LogFile](#) directive sets a destination file for NXLog internal logs. If this directive is unset, the log file is disabled and internal NXLog logs are not written to file (unless configured via the [im_internal](#) module). See also [Rotating the Internal Log File](#).

With the [User](#) and [Group](#) directives set, NXLog will drop root privileges after starting and run under the specified user and group. These directives are ignored if running on the Windows platform.

After starting, NXLog will change its working directory to the directory specified by the [SpoolDir](#) directive. Non-absolute paths in the configuration will be relative to this directory.

See the Reference Manual for a [complete list of available global directives](#).

18.2. Modules

NXLog will only load modules which are specified in the configuration file and used in an active route. The following is a skeleton configuration block for an input module:

nxlog.conf

```
1 <Input instanceName>
2     Module      im_module
3     ...
4 </Input>
```

The instance name must be unique and can contain only the characters [a-zA-Z0-9_-]. The instance name is referenced from the [Route](#) definition. Four types of modules exist in NXLog; these must be declared with the [Input](#), [Processor](#), [Output](#), and [Extension](#) tags.

18.3. Routes

Routes define the flow and processing order of the log messages. A route must have a name and a [Path](#). The name is specified like the instance name of a module block.

Example 10. Route Block

This Route instance, named `example`, takes logs from Input module instances named `in1` and `in2`, processes the logs with the `proc` Processor module instance, and sends the resulting logs to both Output module instances `out1` and `out2`. These named module instances must be defined elsewhere in the configuration file.

nxlog.conf

```
1 <Route example>
2     Path      in1, in2 => proc => out1, out2
3 </Route>
```

If no Route block is specified in the configuration, NXLog will automatically generate a route, with all the Input and Output instances specified in a single path.

Example 11. An Automatic Route Block

NXLog can use a configuration with no Route block, such as the following.

nxlog.conf

```
1 <Input in1>
2     Module im_null
3 </Input>
4
5 <Input in2>
6     Module im_null
7 </Input>
8
9 <Output out1>
10    Module om_null
11 </Output>
12
13 <Output out2>
14    Module om_null
15 </Output>
```

The following Route block will be generated automatically.

nxlog.conf (Generated Route)

```
1 <Route r>
2     Path    in1, in2 => out1, out2
3 </Route>
```

18.4. Constant and Macro Definitions

A **define** is useful if there are many instances in the configuration where the same value must be used. Typically, defines are used for directories and hostnames. In such cases the value can be configured with a single definition. In addition to constants, other strings like code snippets or parser rules can be defined in this way.

An NXLog define works in a similar way to the C language, where the pre-processor substitutes the value in places where the macro is used. The NXLog configuration parser replaces all occurrences of the defined name with its value, and then after this substitution the configuration check occurs.

Example 12. Using Defines

This example shows the use of two defines: `BASEDIR` and `IGNORE_DEBUG`. The first is a simple constant, and its value is used in two `File` directives. The second is an NXLog language statement, it is used in an `Exec` directive.

nxlog.conf

```

1 define BASEDIR /var/log
2 define IGNORE_DEBUG if $raw_event =~ /debug/ drop();
3
4 <Input messages>
5   Module im_file
6   File   '%BASEDIR%/messages'
7 </Input>
8
9 <Input proftpd>
10  Module im_file
11  File   '%BASEDIR%/proftpd.log'
12  Exec   %IGNORE_DEBUG%
13 </Input>
```

The `define` directive can be used for statements as shown above, but multiple statements should be specified using a code `block`, with curly braces (`{}`), to result in the expected behavior.

Example 13. Incorrect Use of a Define

The following example shows an **incorrect** use of the `define` directive. After substitution, the `drop()` procedure will always be executed; only the warning message will be logged conditionally.

nxlog.conf (incorrect)

```

1 define ACTION log_warning("dropping message"); drop();
2
3 <Input in>
4   Module im_file
5   File   '/var/log/messages'
6   Exec   if $raw_event =~ /dropme/ %ACTION%
7 </Input>
```

To avoid this problem, the action should be defined using a code `block`.

```
1 define ACTION { log_warning("dropping message"); drop(); }
```

18.5. Environment Variables

The `envvar` directive works like `define` except that the value is retrieved from the environment. This makes it possible to reference the environment variable as if it was a `define`. This directive is only available in NXLog Enterprise Edition.

Example 14. Using Environment Variables

This is similar to the previous [example using a define](#), but here the value is fetched from the environment.

nxlog.conf

```
1 envvar BASEDIR
2
3 <Input in>
4     Module im_file
5     File   '%BASEDIR%/messages'
6 </Input>
```

18.6. File Inclusion

NXLog provides several features for including configuration directives and blocks from separate files or from executables.

NOTE

The [SpoolDir](#) directive does not take effect until after the configuration has been parsed, so relative paths specified with these directives are relative to the working directory where NXLog was started from. Generally, it is recommended to use absolute paths. If desired, [define](#) directives can be used to simulate relative paths (see [Using Defines to Include a Configuration File](#)).

With the [include](#) directive it is possible to specify a file or set of files to be included in the current configuration file.

Example 15. Including a Configuration File

This example includes the contents of the `/opt/nxlog/etc/syslog.conf` file in the current configuration.

nxlog.conf

```
1 include /opt/nxlog/etc/syslog.conf
```

Example 16. Using Defines to Include a Configuration File

In this example for Windows, two [define](#) directives are used to include an `eventlog.conf` configuration file.

nxlog.conf

```
1 define ROOT C:\Program Files (x86)\nxlog
2 define CONFDIR %ROOT%\conf
3 include %CONFDIR%\eventlog.conf
```

The [include](#) directive also supports wildcarded filenames. A set of files in a directory can be included without the need to explicitly list each one. This could be used to specify a drop-in directory for OS-specific configuration snippets (like `windows2003.conf`) or application-specific snippets (such as `syslog.conf`).

Example 17. Including a Configuration Directory

This example includes all `.conf` files in `/opt/nxlog/etc/nxlog.d`.

nxlog.conf

```
1 include /opt/nxlog/etc/nxlog.d/*.conf
```

With the `include_stdout` directive, an external command can be used to provide configuration content. There are many ways this could be used, including fetching, decrypting, and validating a signed configuration from a remote host, or generating configuration content dynamically.

Example 18. Using an Executable to Generate Configuration

Here, a separate script is responsible for fetching the NXLog configuration.

```
nxlog.conf
1 include_stdout /opt/nxlog/etc/fetch_conf.sh
```

Chapter 19. NXLog Language

The NXLog core has a built-in interpreted language. This language can be used to make complex decisions or build expressions in the NXLog configuration file. Code written in the NXLog language is similar to Perl, which is commonly used by developers and administrators for log processing tasks. When NXLog starts and reads its configuration file, directives containing NXLog language code are parsed and compiled into a pseudo-code. If a syntax error is found, NXLog will print the error. The pseudo-code is then evaluated at run-time, as with other interpreted languages.

The features of the NXLog language are not limited to those in the NXLog core: modules can register functions and procedures to supplement the built-in [functions](#) and [procedures](#) (see [Functions](#), for example).

NOTE

Due to the simplicity of the language there is no error handling (except for function return values) available to the user. If an error occurs during the execution of the NXLog pseudo-code, usually the error is printed in the NXLog logs. If an error occurs during log message processing it is also possible for the message to be dropped. If sophisticated error handling or more complex processing is required, the message processing can be implemented in an external script or program via the [xm_exec](#) module, in a dedicated NXLog module, or in Perl via the [xm_perl](#) module.

The NXLog language is described in five sections.

[Types](#)

All fields and other expressions in the NXLog language are *typed*.

[Expressions](#)

An *expression* is evaluated to a value at run-time and the value is used in place of the expression. All expressions have types. Expressions can be used as arguments for some module directives.

[Statements](#)

The evaluation of a *statement* will cause a change in the state of the NXLog engine, the state of a module instance, or the current event. Statements often contain expressions. Statements are used as arguments for the [Exec](#) module directive, where they are then executed for each event (unless [scheduled](#)).

[Variables](#)

Variables store data persistently in a module instance, across multiple event records.

[Statistical Counters](#)

NXLog provides *statistical counters* with various algorithms that can be used for realtime analysis.

Example 19. Statements vs. Configurations

While this Guide provides many *configuration* examples, in some cases only *statement* examples are given. Statements must be used with the [Exec](#) directive (or Exec block). The following *statement* example shows a way to use the [parsedate\(\)](#) function.

```
1 if $raw_event =~ ^(\w{3} \d{2} \d{2}:\d{2}:\d{2})/
2     $EventTime = parsedate($1);
```

The following *configuration* example uses the above statement in an Exec block.

nxlog.conf

```
1 <Input in>
2     Module im_file
3     File   '/var/log/app.log'
4     <Exec>
5         if $raw_event =~ ^(\w{3} \d{2} \d{2}:\d{2}:\d{2})/
6             $EventTime = parsedate($1);
7     </Exec>
8 </Input>
```

19.1. Types

The NXLog language is a typed language. Fields, literals, and other expressions evaluate to values with specific types. This allows for stricter type-safety syntax checking when parsing the configuration. Note that [fields](#) and some functions can return values with types that can only be determined at run-time.

NOTE The language provides only simple types, complex types such as arrays and hashes (associative arrays) are not supported. The language does support the [undefined](#) value similar to that in Perl. See the [xm_perl](#) module if you require more complex types.

A log format must be parsed before its individual parts can be used for processing (see [Fields](#)). But even after the message has been parsed into its parts, additional processing may still be required to, for example, prepare a timestamp for comparison with another timestamp. This is where typing is helpful: by converting all timestamps to [datetime](#) types, they can be easily compared and later converted to strings if required using provided functions and procedures. The same applies to other types.

Example 20. Typed Fields in a Syslog Event Record

The following input configuration, and list below, show the processing of typed fields in a Syslog event record.

nxlog.conf

```
1 <Input in>
2     # 1. New event record created
3     Module im_udp
4     Host   0.0.0.0
5     Port   514
6 <Exec>
7     # 2. Timestamp parsed from Syslog header
8     if $raw_event =~ /(\w{3} \d{2} \d{2}:\d{2}:\d{2})/
9     {
10        # 3. parsedate() function converts from string to datetime
11        $EventTime = parsedate($1);
12        # 4. Datetime fields compared
13        if ( $EventReceivedTime - $EventTime ) > 60000000
14            log_warning('Message delayed more than 1 minute');
15    }
16 </Exec>
17 </Input>
```

1. NXLog creates a new event record for the incoming log message. The new event record contains the **\$raw_event** string type field, with the contents of the entire Syslog string.
2. A regular expression is used to parse the timestamp from the event. The captured sub-string is a string type, not a datetime type.
3. The **parsedate()** function converts the captured string to a **datetime** type.
4. Two datetime fields are compared to determine if the message was delayed during delivery. The datetime type **\$EventReceivedTime** field is **added by NXLog** to each event when it is received.

NOTE

Normally the **parse_syslog()** procedure (provided by the **xm_syslog** extension module) would be used to parse a Syslog event. It will create **fields** with the appropriate types during parsing, eliminating the need to directly call the **parsedate()** function. See [Collecting and Parsing Syslog](#).

For a full list of types, see the Reference Manual [Types](#) section. For NXLog language core functions that can be used to work with types, see [Functions](#). For functions and procedures that can work with types related to a particular format, see the module corresponding to the required format.

19.2. Expressions

An expression is a language element that is dynamically evaluated to a value at run-time. The value is then used in place of the expression. Each expression evaluates to a type, but not always to the same type.

The following language elements are expressions: [literals](#), [regular expressions](#), [fields](#), [operations](#), and [functions](#).

Expressions can be bracketed by parentheses (()) to help improve code readability.

Example 21. Using Brackets Around Expressions

There are three statements below, one per line. Each statement contains multiple expressions, with parentheses added in various ways.

```
1 if 1 + 1 == (1 + 1) log_info("2");
2 if (1 + 1) == (1 + 1) log_info("2");
3 if ((1 + 1) == (1 + 1)) log_info("2");
```

Expressions are often used in statements.

Example 22. Using an Expression in a Statement

This simple statement uses the [log_info\(\)](#) procedure with an expression as its argument. The expression, in this case, is a *literal*.

```
1 log_info('This message will be logged.');
```

Here is a [function](#) (also an expression) that is used in the same procedure. It generates an internal event with the current time when each event is processed.

```
1 log_info(now());
```

Expressions can be used with module directives that support them.

Example 23. Expressions for Directives

The [File](#) directive of the [om_file](#) module supports expressions. This allows the output filename to be set dynamically for each individual event.

nxlog.conf

```
1 <Output out>
2   Module  om_file
3   File    "/var/log/nxlog/out_" + strftime($EventTime, "%Y%m%d")
4 </Output>
```

See [Using Dynamic Filenames](#) for more information.

19.2.1. Literals

A *literal* is a simple expression that represents a fixed value. Common literals include booleans, integers, and strings. The type of literal is detected by the syntax used to declare it.

NOTE This section demonstrates the use of literals by using examples with [assignment](#) statements.

[Boolean literals](#) are either TRUE or FALSE, and are case-insensitively declared using one of those keywords.

Setting Boolean Literals

```
1 $Important = FALSE;
2 $Local = true;
```

[Integer literals](#) are declared with an unquoted integer. Negative integers, hexadeciml notation, and base-2 modifiers (Kilo, Mega, and Giga) are supported.

Setting Integer Literals

```
1 $Count = 42;
2 $NegativeCount = -42;
3 $BigCount = 42M;
4 $HexCount = 0x2A;
```

[String literals](#) are declared by quoting characters with single or double quotes. Escape sequences are available when using double quotes.

Setting String Literals

```
1 $Server = 'Alpha';
2 $Message = 'This is a test message.';
3 $NumberAsString = '12';
4 $StringWithNewline = "This is line 1.\nThis is line 2.";
```

For a list of all available literals, see the Reference Manual [Literals](#) section.

19.2.2. Regular Expressions

NXLog supports regular expressions for matching, parsing, and modifying event records. In the context of the NXLog language, a regular expression is an expression that is evaluated to a boolean value at run-time. Regular expressions can be used to define complex search and replacement patterns for text matching and substitution.

NOTE

Examples in this section use only simple patterns. See [Extracting Data](#) and other topic-specific sections for more extensive examples.

Matching can be used with an *if* statement to conditionally execute a statement.

Example 24. Matching a Field With a Regular Expression

The event record will be discarded if the `$raw_event` field matches the regular expression.

```
1 if $raw_event =~ /TEST: / drop();
```

Regular expression matching can also be used for extensive parsing, by capturing sub-strings for field assignment.

Example 25. Parsing Fields With a Regular Expression

If the `$raw_event` field contains the regular expression, the two fields will be set to the corresponding captured sub-strings.

```
1 if $raw_event =~ /TEST(\d): (.+)/
2 {
3     $TestNumber = $1;
4     $TestName = $2;
5 }
```

Regular expression substitution can be used to modify a string. In this case, the regular expression follows the form `s/pattern/replace/`. The result of the expression will be assigned to the field to the left of the operator.

Example 26. Performing Substitution Using a Regular Expression

The first regular expression match will be removed from the `$raw_event` field.

```
1 $raw_event =~ s/TEST: //;
```

Global substitution is supported with the `/g` modifier. Without the `/g` modifier, only the first match in the string will be replaced.

Example 27. Global Regular Expression Substitution

Every whitespace character in the `$AlertType` field will be replaced with an underscore (`_`).

```
1 $AlertType =~ s/\s/_/g;
```

A statement can be conditionally executed according to the success of a regular expression substitution.

Example 28. Regular Expression Substitution With Conditional Execution

If the substitution succeeds, an internal log message will also be generated.

```
1 if $Hostname =~ s/myhost/yourhost/ log_info('Updated hostname');
```

For more information, see the following sections in the Reference Manual: [Regular Expressions](#), `=~`, and `!~`.

19.2.3. Fields

When NXLog receives a log message, it creates an event record for it. An event record is a set of *fields* (see [Fields](#) for more information). A field is an expression which evaluates to a value with a specific *type*. Each field has a name, and in the NXLog language it is represented with the dollar sign (\$) prepended to the name of the field, like Perl's scalar variables.

Fields are only available in an evaluation context which is triggered by a log message. For example, using a value of a field in the [Exec directive of a Schedule block](#) will result in a run-time error because the scheduled execution is not triggered by a log message.

Because it is through fields that the NXLog language accesses the contents of an event record, they are frequently referenced. The following examples show some common ways that fields are used in NXLog configurations.

Example 29. Assigning a Value to a Field

This statement uses [assignment](#) to set the `$Department` field on log messages.

```
1 $Department = 'customer-service';
```

Example 30. Testing a Field Value

If the `$Hostname` field does not match, the message will be discarded with the [drop\(\)](#) procedure.

```
1 if $Hostname != 'webserver' drop();
```

Example 31. Using a Field in a Procedure

This statement will generate an internal event if **\$SeverityValue** integer field is greater than **2** (NXLog **INFO** severity). The generated event will include the contents of the **\$Message** field.

```
1 if $SeverityValue > 2 log_warning("ALERT: " + $Message);
```

19.2.4. Operations

Like other programming languages and especially Perl, the NXLog language has unary operations, binary operations, and the conditional ternary operation. These operations are [expressions](#) and evaluate to values.

Unary Operations

Unary operations work with a single operand and evaluate to a boolean value.

Example 32. Using a Unary Operation

This statement uses the **defined** operator to log a message only if the **\$Hostname** field is defined in the event record.

```
1 if defined $Hostname log_info('Event received');
```

Binary Operations

Binary operations work with two operands and evaluate to a value. The type of the evaluated value depends on the type of the operands. Execution might result in a run-time error if the type of the operands are unknown at compile time and then evaluate to types which are incompatible with the binary operation when executed.

Example 33. Using Binary Operations

This statement uses the **==** operator to drop the event if the **\$Hostname** field matches.

```
1 if $Hostname == 'testbox' drop();
```

Here, the **+** operator is used to concatenate two strings.

```
1 log_info('Event received from ' + $Hostname);
```

Ternary Operation

The conditional or [ternary operation](#) requires three operands. The first is an expression that evaluates to a boolean. The second is an expression that is evaluated if the first expression is TRUE. The third is an expression that is evaluated if the first expression is FALSE.

Example 34. Using the Ternary Operation

This statement sets the **\$Important** field to TRUE if **\$SeverityValue** is greater than **2**, or FALSE otherwise. The parentheses are optional and have been added here for clarity.

```
1 $Important = ( $SeverityValue > 2 ? TRUE : FALSE );
```

For a full list of supported operations, see the Reference Manual [Operations](#) section.

19.2.5. Functions

A *function* is an expression which always returns a value. A function cannot be used without using its return value. Functions can be polymorphic: the same function can take different argument types.

Many NXLog language features are provided through functions. As with other types of expressions, and unlike procedures, a function never modifies the state of the NXLog engine, the state of the module, or the current event.

See the list of [core functions](#). Modules can provide additional functions for use with the NXLog language.

Example 35. Function Calls

These statements use the `now()` function (returning the current time) and the `hostname()` function (returning the hostname of the system running NXLog) to set fields.

```
1 $EventTime = now();  
2 $Relay = hostname();
```

Here, any event with a `$Message` field over 4096 bytes causes an internal log to be generated.

```
1 if size($Message) > 4096 log_info('Large message received.');
```

19.3. Statements

The evaluation of a *statement* will usually result in a change in the state of the NXLog engine, the state of a module, or the log message.

Statements are used with the `Exec` module directive. A statement is terminated by a semicolon (`;`).

Example 36. Using a Statement with Exec

With this input configuration, an internal NXLog log message will be generated for each message received.

`nxlog.conf`

```
1 <Input in>  
2     Module  im_udp  
3     Host    0.0.0.0  
4     Port    514  
5     Exec    log_info("Message received on UDP port 514");  
6 </Input>
```

Multiple statements can be specified, these will be evaluated and executed in order. Statements can also be given on multiple lines by using line continuation or by enclosing the statements in an [Exec block](#).

Example 37. Using Multiple Statements with Exec

This configuration generates an internal log message and sets the **\$File** field.

nxlog.conf

```
1 <Input in1>
2   Module im_file
3   File   '/var/log/app.log'
4   Exec   log_info("App message read from log"); $File = file_name();
5 </Input>
```

This is the same, but the backslash (\) is used to continue the **Exec** directive to the next line.

nxlog.conf

```
1 <Input in2>
2   Module im_file
3   File   '/var/log/app.log'
4   Exec   log_info("App message read from log"); \
5           $File = file_name();
6 </Input>
```

This also is the same, but uses an **Exec block** instead. The backslash is not necessary here.

nxlog.conf

```
1 <Input in3>
2   Module im_file
3   File   '/var/log/app.log'
4   <Exec>
5     log_info("App message read from log");
6     $File = file_name();
7   </Exec>
8 </Input>
```

Statements can also be executed based on a schedule, by using the **Exec** directive of a **Schedule** block. The **Exec** directive is slightly different in this case: because its execution is based on a schedule rather than a log event, there is no associated event record. The **\$File** field assignment in the example above would be impossible.

Example 38. Using a Statement in a Schedule

This input instance will generate an hourly internal log event.

nxlog.conf

```
1 <Input syslog_udp>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Schedule>
6     When   @hourly
7     Exec   log_info("The syslog_udp input module instance is active.");
8   </Schedule>
9 </Input>
```

NOTE

Similar functionality is implemented by the **im_mark** module.

19.3.1. Assignment

Each event record is made up of fields, and *assignment* is the primary way that a value is written to a field in the NXLog language. The assignment operation is declared with an equal sign (`=`). This operation loads the value from the expression evaluated on the right into an event record `field` on the left.

Example 39. Using Field Assignment

This input instance uses assignment operations to add two fields to each event record.

`nxlog.conf`

```
1 <Input in>
2   Module im_file
3   File   '/var/log/messages'
4   <Exec>
5     $Department = 'processing';
6     $Tier = 1;
7   </Exec>
8 </Input>
```

19.3.2. Block

Statements can be declared inside a *block* by surrounding them with curly braces (`{}`). A statement block in the configuration is parsed as if it were a single statement. Blocks are typically used with [conditional statements](#).

Example 40. Using Statement Blocks

This statement uses a block to execute two statements if the `$Message` field matches.

`nxlog.conf`

```
1 <Input in>
2   Module im_file
3   File   '/var/log/messages'
4   <Exec>
5     if $Message =~ /^br0:/{
6       {
7         log_warning('br0 interface state changed');
8         $Tag = 'network';
9       }
10    </Exec>
11 </Input>
```

19.3.3. Procedures

While functions are expressions that evaluate to values, *procedures* are statements that perform actions. Both functions and procedures can take arguments. Unlike functions, procedures never return values. Instead, a procedure modifies its argument, the state of the NXLog engine, the state of a module, or the current event. Procedures can be polymorphic: the same procedure can take different argument types.

Many NXLog language features are provided through procedures. See the [list of available procedures](#). Modules can provide additional procedures for use with the NXLog language.

Example 41. Using a Procedure

This example uses the `parse_syslog()` procedure, provided by the `xm_syslog` module, to parse each Syslog-formatted event record received via UDP.

`nxlog.conf`

```
1 <Input in>
2     Module im_udp
3     Host   0.0.0.0
4     Port   514
5     Exec   parse_syslog();
6 </Input>
```

19.3.4. If-Else

The `if` or *conditional* statement allows a statement to be executed based on the boolean value of an expression. When the boolean is TRUE, the statement is executed. An optional `else` keyword can be followed by another statement to be executed if the boolean is FALSE.

Example 42. Using If Statements

This example uses an *if* statement and the [drop\(\)](#) procedure to discard any event that matches the regular expression.

nxlog.conf

```
1 <Input in1>
2   Module im_file
3   File   '/var/log/messages'
4   Exec   if $raw_event =~ /junk/ drop();
5 </Input>
```

Here, any event *not* matching the regular expression will be dropped.

nxlog.conf

```
1 <Input in2>
2   Module im_file
3   File   '/var/log/messages'
4   Exec   if not ($raw_event =~ /important/) drop();
5 </Input>
```

Finally, this statement shows more extensive use of the *if* statement, with an *else* clause and [blocks](#) defined by curly braces (`{}`).

nxlog.conf

```
1 <Input in3>
2   Module im_file
3   File   '/var/log/messages'
4   <Exec>
5     if $raw_event =~ /alert/
6     {
7       log_warning('Detected alert message');
8     }
9   else
10   {
11     log_info('Discarding non-alert message');
12     drop();
13   }
14 </Exec>
15 </Input>
```

19.4. Variables

While NXLog provides [fields](#) for storing data during the processing of an event, they are only available for the duration of that event record and can not be used to store a value across multiple events. For this purpose, module *variables* can be used. A variable stores a value for the module instance where it is set. It can only be accessed from the same module where it was created: a variable with the same name is a different variable when referenced from another module.

Each module variable can be created with an expiry value or an infinite lifetime. If an expiry is used, the variable will be destroyed automatically when the lifetime expires. This can be used as a garbage collection method or to reset variable values automatically.

A module variable is referenced by a string value and can store a value of any type. Module variables are supported by all modules. See the [create_var\(\)](#), [delete_var\(\)](#), [set_var\(\)](#), and [get_var\(\)](#) procedures.

Example 43. Using Module Variables

If the number of login failures exceeds 3 within 45 seconds, then an internal log message is generated.

nxlog.conf

```

1 <Input in>
2   Module im_file
3   File   '/var/log/messages'
4   <Exec>
5   if $Message =~ /login failure/
6   {
7     if not defined get_var('login_failures')
8     { # create the variable if it doesn't exist
9       create_var('login_failures', 45);
10      set_var('login_failures', 1);
11    }
12    else
13    { # increase the variable and check if it is over the limit
14      set_var('login_failures', get_var('login_failures') + 1);
15      if get_var('login_failures') >= 3
16        log_warning(">= 3 login failures within 45 seconds");
17    }
18  }
19 </Exec>
20 </Input>
```

NOTE

The [pm_evcorr](#) module is recommended instead for this case. This algorithm does not reliably detect failures because the lifetime of the variable is not affected by [set_var\(\)](#). For example, consider login failures at 0, 44, 46, and 47 seconds. The lifetime of the variable will be set when the first failure occurs, causing the variable to be cleared at 45 seconds. The variable is created with a new expiry at 46 seconds, but then only two failures are noticed. Also, this method can only work in real-time because the timing is not based on values available in the log message (though the event time could be stored in another variable).

19.5. Statistical Counters

Like [variables](#), *statistical counters* provide data storage for a module instance. Counters only support integers, but a counter can use an [algorithm](#) to recalculate its value every time it is updated or read. A statistical counter will only return a value if the time specified in the interval argument has elapsed since it was created. Statistical counters can also be created with a lifetime. When a counter expires, it is destroyed, like module variables.

A statistical counter can be created with the [create_stat\(\)](#) procedure call. After it is created, it can be updated with the [add_stat\(\)](#) procedure call. The value of the counter can be read with the [get_stat\(\)](#) function call. Note that the value of the statistical counter is only recalculated during these calls, rather than happening automatically. This can result in some slight distortion of the calculated value if the add and read operations are infrequent.

A time value can also be specified during [creation](#), [updating](#), and [reading](#). This makes it possible for statistical counters to be used with offline log processing.

Example 44. Using Statistical Counters

This input configuration uses a **Schedule** block and a statistical counter with the **RATEMAX** algorithm to calculate the maximum rate of events over a 1 hour period. An internal log message is generated if the rate exceeds 500 events/second at any point during the 1 hour period.

nxlog.conf

```
1 <Input in>
2   Module im_tcp
3   Host   0.0.0.0
4   Port   1514
5   <Exec>
6     parse_syslog();
7     if defined get_stat('eps') add_stat('eps', 1, $EventReceivedTime);
8   </Exec>
9   <Schedule>
10    Every 1 hour
11    <Exec>
12      create_stat('eps', 'RATEMAX', 1, now(), 3600);
13      if get_stat('eps') > 500
14        log_info('Inbound TCP rate peaked at ' + get_stat('eps')
15                      + ' events/second during the last hour');
16    </Exec>
17  </Schedule>
18 </Input>
```

Chapter 20. Reading and Receiving Logs

This chapter discusses log sources that you may need to use with NXLog, including:

- log data received over the [network](#),
- events stored in [databases](#),
- messages read from [files](#), and
- data retrieved using [executables](#).

20.1. Receiving over the Network

This section provides information and examples about receiving log messages from the network over various protocols.

UDP

The [im_udp](#) module handles incoming messages over UDP.

Example 45. Using the im_udp Module

This input module instance shows the [im_udp](#) module configured with the default options: localhost only and port 514.

nxlog.conf

```
1 <Input udp>
2   Module  im_udp
3   Host    localhost
4   Port    514
5 </Input>
```

The UDP protocol does not guarantee reliable message delivery. It is recommended to use the TCP or SSL transport modules instead if message loss is a concern.

NOTE

Though NXLog was designed to minimize message loss even in the case of UDP, adjusting the kernel buffers may reduce the likelihood of UDP message loss on a loaded system. The [Priority](#) directive in the Route block can also help.

TCP

The [im_tcp](#) module handles incoming messages over TCP. For TLS/SSL, use the [im_ssl](#) module.

Example 46. Using the im_tcp Module

This input module instance accepts TCP connections from anywhere on port 1514.

nxlog.conf

```
1 <Input tcp>
2   Module  im_tcp
3   Host    0.0.0.0
4   Port    1514
5 </Input>
```

SSL/TLS

The [im_ssl](#) module handles incoming messages over TCP with SSL/TLS security.

Example 47. Using the `im_ssl` Module

The following input module instance listens for SSL/TLS encrypted incoming logs on port 6514. The certificate file paths are specified relative to a previously defined `CERTDIR`.

`nxlog.conf`

```
1 <Input in>
2   Module      im_ssl
3   Host        0.0.0.0
4   Port        6514
5   CAFile      %CERTDIR%/ca.pem
6   CertFile    %CERTDIR%/client-cert.pem
7   CertKeyFile %CERTDIR%/client-key.pem
8 </Input>
```

Syslog

To receive Syslog over the network, use one of the network modules above, coupled with `xm_syslog`. Syslog parsing is not required if you only need to forward or store the messages as they are. See also [Accepting Syslog via UDP, TCP, or TLS](#).

Example 48. Receiving Syslog over TCP with Octet-Framing

With this example configuration, NXLog listens for messages on TCP port 1514. The `xm_syslog` extension module provides the `Syslog_TLS` InputType (for octet-framing) and the `parse_syslog()` procedure for parsing Syslog messages.

`nxlog.conf`

```
1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input in>
6   Module      im_tcp
7   Host        0.0.0.0
8   Port        1514
9   # "Syslog_TLS" is for octet framing and may be used with TLS/SSL
10  InputType   Syslog_TLS
11  Exec        parse_syslog();
12 </Input>
```

20.2. Reading from a Database

With the `im_dbi` and `im_odbc` modules it is possible to read logs directly from database servers. The `im_dbi` module can be used on POSIX systems where libdbi is available. The `im_odbc` module, available in NXLog Enterprise Edition, can be used with ODBC compatible databases on Windows, Linux, and Unix.

Example 49. Using the im_dbi Module

This example uses libdbi and the MySQL driver to insert records into the **logdb** database.

nxlog.conf

```
1 <Input in>
2   Module  im_dbi
3   Driver   mysql
4   Option   host 127.0.0.1
5   Option   username mysql
6   Option   password mysql
7   Option   dbname logdb
8   SQL      SELECT id, facility, severity, hostname, timestamp, application, \
9                   message FROM log
10 </Input>
```

Example 50. Using the im_odbc Module

Here, the **mydb** database is accessed via ODBC.

nxlog.conf

```
1 <Input in>
2   Module      im_odbc
3   ConnectionString DSN=mssql;database=mydb;
4   SQL         SELECT RecordNumber as d, DateOccured as EventTime, \
5                   data as Message from logtable WHERE RecordNumber > ?
6 </Input>
```

20.3. Reading from Files and Sockets

Files

The **im_file** module can be used to read logs from files. See also [Reading Syslog Log Files](#).

Example 51. Using the im_file Module

This example reads from the specified file without performing any additional processing.

nxlog.conf

```
1 <Input in>
2   Module  im_file
3   File    "/var/log/messages"
4 </Input>
```

Unix Domain Socket

Use the **im_uds** module to read from a Unix domain socket. See also [Accepting Syslog via /dev/log](#).

Example 52. Using the im_uds Module

With this configuration, NXLog will read messages from the `/dev/log` socket. NXLog's flow control feature must be disabled in this case (see the [FlowControl](#) directive in the Reference Manual).

nxlog.conf

```
1 <Input in>
2     Module      im_uds
3     UDS        /dev/log
4     FlowControl FALSE
5 </Input>
```

20.4. Receiving from an Executable

The [im_exec](#) module can be used to read logs from external programs and scripts over a pipe.

Example 53. Using the im_exec Module

This example uses the tail command to read messages from a file.

NOTE

The [im_file](#) module should be used to read log messages from files. This example only demonstrates the use of the [im_exec](#) module.

nxlog.conf

```
1 <Input in>
2     Module    im_exec
3     Command   /usr/bin/tail
4     Arg       -f
5     Arg       /var/log/messages
6 </Input>
```

Chapter 21. Processing Logs

This chapter deals with various tasks that might be required after a log message is received by NXLog.

21.1. Parsing Various Formats

After an input module has received a log message and generated an event record for it, there may be additional parsing required. This parsing can be implemented by a dedicated module, or in the NXLog language with regular expression and other string manipulation functionality.

The following sections provide configuration examples for parsing log formats commonly used by applications.

21.1.1. Common & Combined Log Formats

The Common Log Format (or NCSA Common Log Format) and Combined Log Format are access log formats used by web servers. These are the same, except that the Combined Log Format uses two additional fields.

Common Log Format Syntax

```
host ident authuser [date] "request" status size
```

Combined Log Format Syntax

```
host ident authuser [date] "request" status size "referer" "user-agent"
```

If a field is not available, a hyphen (-) is used as a placeholder.

Table 48. Fields

Field	Description
host	IP address of the client
ident	RFC 1413 identity of the client
authuser	Username of the user accessing the document (not applicable for public documents)
date	Timestamp of the request
request	Request line received from the client
status	HTTP status code returned to the client
size	Size of the object returned to the client (measured in bytes)
referer	URL from which the user was referred
user-agent	User agent string sent by the client

Example 54. Parsing the Common Log Format

This configuration uses a regular expression to parse the fields in each record. The [parsedate\(\)](#) function is used to convert the timestamp string into a [datetime](#) type for later processing or conversion as required.

nxlog.conf

```
<Input access_log>
  Module im_file
  File  "/var/log/apache2/access.log"
  <Exec>
    if $raw_event =~ /(?x)^(\S+)\ \S+\ (\S+)\ \[(\[\^\]]+)\]\ \"(\S+)\ (\.+)
      \ HTTP\/\d\.\d\" (\S+)\ (\S+)/
    {
      $Hostname = $1;
      if $2 != '-' $AccountName = $2;
      $EventTime = parsedate($3);
      $HTTPMethod = $4;
      $HTTPURL = $5;
      $HTTPResponseStatus = $6;
      if $7 != '-' $FileSize = $7;
    }
  </Exec>
</Input>
```

Example 55. Parsing the Combined Log Format

This example is like the previous, but parses the additional two fields in the Combined Log Format. An `om_file` instance is also shown here: it discards all events not related to the user `john` and writes the remaining events to a file in JSON format.

nxlog.conf

```
<Extension _json>
    Module xm_json
</Extension>

<Input access_log>
    Module im_file
    File   "/var/log/apache2/access.log"
    <Exec>
        if $raw_event =~ /(?x)^(\S+)\ \S+\ (\S+)\ \[(\[\^\"]+)\]\ \"(\S+)\ (.+)
                    \ HTTP\/\d.\d\"(\S+)\ (\S+)\ \"(\[\^\"]+)\"
                    \ \"(\[\^\"]+)\"
{
    $Hostname = $1;
    if $2 != '-' $AccountName = $2;
    $EventTime = parsedate($3);
    $HTTPMethod = $4;
    $HTTPURL = $5;
    $HTTPResponseStatus = $6;
    if $7 != '-' $FileSize = $7;
    if $8 != '-' $HTTPReferer = $8;
    if $9 != '-' $HTTPUserAgent = $9;
}
</Exec>
</Input>

<Output out>
    Module om_file
    File   '/var/log/john_access.log'
    <Exec>
        if not (defined($AccountName) and ($AccountName == 'john')) drop();
        to_json();
    </Exec>
</Output>
```

For information about using the Common and Combined Log Formats with the Apache HTTP Server, see [Apache HTTP Server](#).

21.1.2. Field Delimited Formats (CSV)

Comma-, space-, or semicolon-separated field list formats are frequently used. The `xm_csv` module can both generate and parse these formats. Multiple `xm_csv` instances can be used to reorder, add, remove, or modify fields before outputting to a different CSV format.

Example 56. Complex CSV Format Conversion

This example reads from the input file and parses it with the `csv1` instance. The `$date` field is then set to the current time and the `$number` field is set to 0 if it is not already defined. Finally, the `csv2` instance is used to generate output with the additional `date` field, a different delimiter, and a different field order.

nxlog.conf

```

1 <Extension csv1>
2   Module      xm_csv
3   Fields      $id, $name, $number
4   FieldTypes  integer, string, integer
5   Delimiter   ,
6 </Extension>
7
8 <Extension csv2>
9   Module      xm_csv
10  Fields      $id, $number, $name, $date
11  Delimiter   ;
12 </Extension>
13
14 <Input filein>
15  Module      im_file
16  File        "/tmp/input"
17  <Exec>
18    csv1->parse_csv();
19    $date = now();
20    if not defined $number $number = 0;
21    csv2->to_csv();
22  </Exec>
23 </Input>
24
25 <Output fileout>
26  Module      om_file
27  File        "/tmp/output"
28 </Output>
```

Input Sample

```

1, "John K.", 42
2, "Joe F.", 43
```

Output Sample

```

1;42;"John K.";2011-01-15 23:45:20
2;43;"Joe F.";2011-01-15 23:45:20
```

21.1.3. JSON

The `xm_json` module provides procedures for generating and parsing log data in JSON format.

Example 57. Using the xm_json Module for Parsing JSON

This example reads JSON-formatted messages from file with the [im_file](#) module. Then the [parse_json\(\)](#) procedure is used to parse the data, setting each JSON field to a field in the event record.

nxlog.conf

```
1 <Extension xm_json>
2   Module xm_json
3 </Extension>
4
5 <Input in>
6   Module im_file
7   File   "/var/log/app.json"
8   Exec   parse_json();
9 </Input>
```

Example 58. Using the xm_json Module for Generating JSON

Here, the [to_json\(\)](#) procedure is used to write all the event record fields to [\\$raw_event](#) in JSON format. This is then written to file using the [om_file](#) module.

nxlog.conf

```
1 <Extension xm_json>
2   Module xm_json
3 </Extension>
4
5 <Output out>
6   Module om_file
7   File   "/var/log/json.log"
8   Exec   to_json();
9 </Output>
```

21.1.4. W3C Extended Log File Format

See the [specification draft](#) of the W3C format. The dedicated [xm_w3c](#) parser module can be used to process W3C formatted logs. See also the [W3C](#) section in the Microsoft IIS chapter.

Log Sample

```
#Version: 1.0
#Date: 2011-07-01 00:00:00
#Fields: date time cs-method cs-uri
2011-07-01 00:34:23 GET /foo/bar1.html
2011-07-01 12:21:16 GET /foo/bar2.html
2011-07-01 12:45:52 GET /foo/bar3.html
2011-07-01 12:57:34 GET /foo/bar4.html
```

Example 59. Parsing W3C Format With xm_w3c

This configuration reads the W3C format log file and parses it with the [xm_w3c](#) module. The fields in the event record are converted to JSON and the logs are forwarded via TCP.

nxlog.conf

```
1 <Extension _json>
2     Module      xm_json
3 </Extension>
4
5 <Extension w3c_parser>
6     Module      xm_w3c
7 </Extension>
8
9 <Input w3c>
10    Module     im_file
11    File       '/var/log/httpd-log'
12    InputType  w3c_parser
13 </Input>
14
15 <Output tcp>
16    Module     om_tcp
17    Host       192.168.12.1
18    Port       1514
19    Exec       to_json();
20 </Output>
```

The W3C format can also be parsed with the [xm_csv](#) module if using NXLog Community Edition.

Example 60. Parsing W3C Format With xm_csv

The following configuration reads a W3C file and tokenizes it with the CSV parser. Header lines starting with a leading hash mark (#) are ignored. The **\$EventTime** field is set from the parsed **date** and **time** fields.

NOTE

The fields in the **xm_csv** module instance below must be updated to correspond with the fields in the W3C file to be parsed.

nxlog.conf

```

1 <Extension w3c_parser>
2   Module      xm_csv
3   Fields      $date, $time, $HTTPMethod, $HTTPURL
4   FieldTypes  string, string, string, string
5   Delimiter   '
6   EscapeChar  '''
7   QuoteChar   '''
8   EscapeControl FALSE
9   UndefValue  -
10 </Extension>
11
12 <Extension _json>
13   Module      xm_json
14 </Extension>
15
16 <Input w3c>
17   Module      im_file
18   File        '/var/log/httpd-log'
19   <Exec>
20     if $raw_event =~ /^#/ drop();
21     else
22     {
23       w3c_parser->parse_csv();
24       $EventTime = parsedate($date + " " + $time);
25     }
26   </Exec>
27 </Input>
```

21.1.5. XML

The **xm_xml** module can be used for generating and parsing structured data in XML format.

Example 61. Using the xm_xml Module for Parsing XML

This configuration uses the **im_file** module to read from file. Then the **parse_xml()** procedure parses the XML into fields in the event record.

nxlog.conf

```

1 <Extension _xml>
2   Module    xm_xml
3 </Extension>
4
5 <Input in>
6   Module   im_file
7   File    "/var/log/app.xml"
8   Exec    parse_xml();
9 </Input>
```

Example 62. Using the xm_xml Module for Generating XML

Here, the fields in the event record are used by the `to_xml()` procedure to generate XML, which is then written to file by the `om_file` module.

`nxlog.conf`

```
1 <Extension _xml>
2     Module  xm_xml
3 </Extension>
4
5 <Output out>
6     Module  om_file
7     File    "/var/log/logs.xml"
8     Exec    to_xml();
9 </Output>
```

21.2. Alerting

NXLog can be configured to generate alerts when specific conditions are met. Here are several ways alerting could be implemented.

21.2.1. Sending Messages to an External Program

The `om_exec` module can pipe messages to an external program or script, which will be started when the `om_exec` module is started. The script is expected to continue running until the `om_exec` module is stopped and the pipe is closed. This functionality can be used for alerting.

Example 63. Using om_exec,om_exec with an External Alerter

In this example Output, all messages not matching the regular expression are dropped, and remaining messages are piped to a custom *alerter*.

`nxlog.conf`

```
1 <Output out>
2     Module  om_exec
3     Command /usr/local/sbin/alerter
4     Arg    -
5     Exec    if not ($raw_event =~ /alertcondition/) drop();
6 </Output>
```

Without the `Exec` directive above, all messages received by the module would be passed to the *alerter*.

See also [Sending to Executables](#).

21.2.2. Invoking a Program for Each Message

The `xm_exec` module provides two procedures, `exec()` and `exec_async()`, for spawning an external program or script. The script is executed once for each call, and is expected to terminate when it has finished processing the message.

Example 64. Using xm_exec with an External Alerter

In this example Input, each message matching the regular expression is piped to a new instance of *alerter*, which is executed asynchronously (does not block additional processing by the calling module).

nxlog.conf

```
1 <Extension _exec>
2     Module  xm_exec
3 </Extension>
4
5 <Input in>
6     Module  im_tcp
7     Host    0.0.0.0
8     Port    1514
9     <Exec>
10    if $raw_event =~ /alertcondition/
11        exec_async("/usr/local/sbin/alerter");
12    </Exec>
13 </Input>
```

Example 65. Using xm_exec to Send an Email

In this example, an email is sent using `exec_async()` when the regular expression condition is met.

nxlog.conf

```
1 <Extension _exec>
2     Module  xm_exec
3 </Extension>
4
5 <Input in>
6     Module  im_tcp
7     Host    0.0.0.0
8     Port    1514
9     <Exec>
10    if $raw_event =~ /alertcondition/
11    {
12        exec_async("/bin/sh", "-c", 'echo "' + $Hostname + '\n\nRawEvent:\n' +
13                    $raw_event + '"|/usr/bin/mail ' +
14                    '-a "Content-Type: text/plain; charset=UTF-8" ' +
15                    '-s "ALERT" user@domain.com');
16    }
17    </Exec>
18 </Input>
```

21.2.3. Generate an Internal NXLog Log Message

NXLog can be configured to generate an internal log event when a specific condition is met. Internal log events can be generated with various severity levels using the `log_error()`, `log_warning()`, `log_info()`, and `log_debug()` procedures. Internal log messages will be written to the file specified by the global `LogFile` directive (according to the configured `LogLevel`) and will be generated by the `im_internal` module.

NOTE

DEBUG level events are not generated by the `im_internal` module.

Example 66. Using log_warning() for Alerting

If a message matches the regular expression, an internal log event is generated with level WARNING.

nxlog.conf

```
1 <Input in>
2   Module im_file
3   File  "/var/log/app.log"
4   Exec  if $raw_event =~ /alertcondition/ log_warning("ALERT");
5 </Input>
```

21.3. Buffering

Each input module has its own read buffer which is filled with data when the source is read. Each processor and output has a limited queue where the route's preceding module places log messages. The default limit for these internal queues is 100 events. For buffering a larger amount of data, the [pm_buffer](#) module provides disk- and memory-based buffering.

Example 67. Using a Memory Buffer to Protect Against UDP Message Loss

In this configuration, a buffer is used for incoming UDP logs to avoid message loss in the case of the TCP output blocking.

nxlog.conf

```
1 <Input udp>
2   Module    im_udp
3   Host     0.0.0.0
4   Port     514
5 </Input>
6
7 <Processor buffer>
8   Module    pm_buffer
9   # 1 MB buffer
10  MaxSize   1024
11  Type      Mem
12  # warn at 512k
13  WarnLimit 512
14 </Processor>
15
16 <Output tcp>
17  Module    om_tcp
18  Host     192.168.1.1
19  Port     1514
20 </Output>
21
22 <Route udp_to_tcp>
23  Path      udp => buffer => tcp
24 </Route>
```

21.4. Character Set Conversion

It is recommended to normalize logs to UTF-8. The [xm_charconv](#) module provides character set conversion: the [convert_fields\(\)](#) procedure for converting an entire message (all event fields) and a [convert\(\)](#) function for converting a string.

Example 68. Character Set Auto-Detection of Various Input Encodings

This configuration shows an example of character set auto-detection. The input file can contain differently encoded lines, and the module normalizes output to UTF-8.

nxlog.conf

```
1 <Extension _charconv>
2   Module      xm_charconv
3   AutodetectCharsets utf-8, euc-jp, utf-16, utf-32, iso8859-2
4 </Extension>
5
6 <Input filein>
7   Module      im_file
8   File        "tmp/input"
9   Exec        convert_fields("auto", "utf-8");
10 </Input>
11
12 <Output fileout>
13   Module      om_file
14   File        "tmp/output"
15 </Output>
16
17 <Route r>
18   Path        filein => fileout
19 </Route>
```

21.5. Detecting a Dead Agent or Log Source

It is a common requirement to detect conditions when there are no log messages coming from a source. This usually indicates a problem such as a broken network connection, a server down, or a stuck application or system service. Usually this problem should be detected by monitoring tools (such as Nagios or OpenView), but the absence of logs can also be a good reason to investigate.

NOTE

The [im_mark](#) module is related: it can generate messages periodically in order to show that the agent is still functioning.

The solution to this problem is the combined use of [statistical counters](#) and Scheduled checks. The input module can update a statistical counter configured to calculate events per hour. In the same input module a [Schedule](#) block checks the value of the statistical counter periodically. When the event rate is zero or drops below a certain limit, an appropriate action can be executed such as sending out an alert email or generating an internal warning message. Note that there are other ways to solve this issue and this method may not be optimal for all situations.

Example 69. Alerting on Absence of Log Messages

The following configuration example creates a statistical counter in the context of the `im_tcp` module to calculate the number of events received per hour. The `Schedule` block within the context of the same module checks the value of the `msgrate` statistical counter and generates an internal error message when there were no logs received in the past hour.

`nxlog.conf`

```
1 <Input in>
2   Module  im_tcp
3   Port    2345
4 <Exec>
5     create_stat("msgrate", "RATE", 3600);
6     add_stat("msgrate", 1);
7 </Exec>
8 <Schedule>
9   Every  3600 sec
10  <Exec>
11    create_stat("msgrate", "RATE", 10);
12    add_stat("msgrate", 0);
13    if defined get_stat("msgrate") and get_stat("msgrate") <= 1
14      log_error("No messages received from the source!");
15  </Exec>
16 </Schedule>
17 </Input>
```

21.6. Event Correlation

It is possible to write correlation rules in the NXLog language using the built-in features such as the variables and statistical counters. While these are quite powerful, some cases cannot be detected with these, especially those conditions which require a sliding window.

A dedicated NXLog module, `pm_evcrr`, is available for advanced correlation requirements. It provides features similar to those of `SEC` and greatly enhances the correlation capabilities of NXLog.

Example 70. Correlation Rules

This following configuration sample contains a rule for each type.

nxlog.conf

```
1 <Processor evcorr>
2   Module          pm_evcorr
3   TimeField      EventTime
4
5   <Simple>
6     Exec           if $Message =~ /^simple/ $raw_event = "got simple";
7   </Simple>
8
9   <Suppressed>
10    # Match input event and execute an action list, but ignore the following
11    # matching events for the next t seconds.
12    Condition      $Message =~ /^suppressed/
13    Interval       30
14    Exec           $raw_event = "suppressing..";
15  </Suppressed>
16
17  <Pair>
18    # If TriggerCondition is true, wait Interval seconds for RequiredCondition
19    # to be true and then do the Exec. If Interval is 0, there is no window on
20    # matching.
21    TriggerCondition $Message =~ /^pair-first/
22    RequiredCondition $Message =~ /^pair-second/
23    Interval         30
24    Exec             $raw_event = "got pair";
25  </Pair>
26
27  <Absence>
28    # If TriggerCondition is true, wait Interval seconds for RequiredCondition
29    # to be true. If RequiredCondition does not become true within the specified
30    # interval then do the Exec.
31    TriggerCondition $Message =~ /^absence-trigger/
32    RequiredCondition $Message =~ /^absence-required/
33    Interval         10
34    Exec             log_info("'absence-required' not received within 10s");
35  </Absence>
36
37  <Thresholded>
38    # If the number of events exceeds the given threshold within the interval do
39    # the Exec. Same as SingleWithThreshold in SEC.
40    Condition      $Message =~ /^thresholded/
41    Threshold       3
42    Interval       60
43    Exec           $raw_event = "got thresholded";
44  </Thresholded>
45
46  <Stop>
47    Condition      $EventTime < 2010-01-02 00:00:00
48    Exec           log_debug("got stop");
49  </Stop>
50
51  <Simple>
52    # This will be rewritten only if the previous Stop condition is FALSE.
53    Exec           $raw_event = "rewritten";
54  </Simple>
55
56 </Processor>
```

21.7. Explicit Drop

NXLog does not drop messages voluntarily. The built-in flow control mechanism ensures that the input modules will pause until the output modules can write. This can be problematic in some situations when dropping messages is preferable to blocking. For this case, the `drop()` procedure can be used in conjunction with the `pm_buffer` module.

Example 71. Using drop() with pm_buffer

This example sends the UDP input to two outputs, a file and a TCP destination. This could be done with a basic single-route configuration such as `Path udp => tcp, file`. However, if the TCP transmission is slower than the rate of incoming UDP packets or the TCP connection is down, the whole chain would be blocked. This would result in dropped UDP packets in the Input `udp` instance. In this situation it is better to selectively drop log messages in the TCP route than to lose them entirely.

The following configuration uses two routes, one for each output. In the TCP route, a `pm_buffer` module is used and the size of the buffer is checked. If the buffer size goes over a certain limit, it is assumed that the TCP output is blocked (or too slow), and messages are explicitly dropped with the `drop()` procedure. In this way the UDP input will not get paused and all messages will be written to the output file regardless of the state of the TCP connection.

nxlog.conf

```
1 <Processor buffer>
2   Module      pm_buffer
3   WarnLimit   800
4   MaxSize     1000
5   Type        Mem
6   Exec        if buffer_size() >= 80k drop();
7 </Processor>
8
9 <Input udp>
10  Module     im_udp
11  Host       0.0.0.0
12  Port       1514
13 </Input>
14
15 <Output tcp>
16  Module     om_tcp
17  Host       192.168.1.1
18  Port       1515
19 </Output>
20
21 <Output file>
22  Module     om_file
23  File       'out.txt'
24 </Output>
25
26 <Route udp_to_tcp>
27  Path       udp => buffer => tcp
28 </Route>
29
30 <Route udp_to_file>
31  Path       udp => file
32 </Route>
```

21.8. Extracting Data

When NXLog receives an event, it creates an event record with a `$raw_event` field, [other core fields](#) like `$EventReceivedTime`, and any fields provided by the particular module (see [Fields](#) for more information). This section explores the various ways that NXLog can be configured to extract values from the raw event.

Some log sources (like Windows EventLog collected via [im_msvisalog](#)) already contain structured data. In this case, there is often no additional extraction required; see [Message Classification](#).

21.8.1. Parser Procedures

NXLog includes many parser procedures that will automatically extract values from a raw log event and create corresponding fields in the event record. These procedures are provided by extension modules. For example, the [xm_syslog](#) module provides the [parse_syslog\(\)](#) procedure, which will parse a BSD or IETF Syslog formatted raw event and create fields in the event record. Other procedures will parse [JSON](#), [key-value pairs](#), [XML](#), and more; many of the available [extension modules](#) provide parser procedures.

Example 72. Parsing a Syslog Event With parse_syslog()

This example shows a Syslog event as it is received via UDP and processed by the [parse_syslog\(\)](#) procedure.

Syslog Message

```
<38>Nov 22 10:30:12 myhost sshd[8459]: Failed password for invalid user linda from 192.168.1.60
port 38176 ssh2←
```

The following configuration loads the [xm_syslog](#) extension module and then uses the [Exec](#) directive to execute the [parse_syslog\(\)](#) procedure for each event.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input udp>
6   Module im_udp
7   Host 0.0.0.0
8   Port 514
9   Exec parse_syslog();
10 </Input>
```

This results in the following fields being added to the event record by [parse_syslog\(\)](#).

Table 49. Syslog Fields Added by parse_syslog()

Field	Value
\$Message	Failed password for invalid user linda from 192.168.1.60 port 38176 ssh2
\$SyslogSeverityValue	6
\$SyslogSeverity	INFO
\$SeverityValue	2
\$Severity	INFO
\$SyslogFacilityValue	4
\$SyslogFacility	AUTH
\$EventTime	2016-11-22 10:30:12
\$Hostname	myhost
\$SourceName	sshd
\$ProcessID	8459

21.8.2. Regular Expressions via the Exec Directive

NXLog supports the use of regular expressions for parsing fields. For detailed information about regular expressions in NXLog, see the Reference Manual [Regular Expressions](#) section.

*Example 73. Parsing With Regular Expressions**Syslog Message*

```
<38>Nov 22 10:30:12 myhost sshd[8459]: Failed password for invalid user linda from 192.168.1.60
port 38176 ssh2<
```

With this configuration, the above Syslog message is first parsed with [parse_syslog\(\)](#). This results in a **\$Message** field created in the event record. Then, a regular expression is used to further parse the **\$Message** field and create additional fields if it matches.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input udp>
6   Module im_udp
7   Host   0.0.0.0
8   Port   514
9   <Exec>
10    parse_syslog();
11    if $Message =~ /(?x)^Failed\ (\S+)\ for(?:\ invalid user)?\ (\S+)\ from
12      \ (\S+)\ port\ \d+\ ssh2$/
13    {
14      $AuthMethod = $1;
15      $AccountName = $2;
16      $SourceIPAddress = $3;
17    }
18  </Exec>
19 </Input>
```

Named capturing is supported also.

nxlog.conf

```
1 <Input in>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Exec>
6     parse_syslog();
7     $Message =~ /(?x)^Failed\ (?<AuthMethod>\S+)\ for(?:\ invalid\ user)?
8       \ (?<AccountName>\S+)\ from\ (?<SourceIPAddress>\S+)\ port
9       \ \d+\ ssh2$/;
10  </Exec>
11 </Input>
```

Table 50. Additional Fields Parsed by Regular Expression

Field	Value
\$AuthMethod	password
\$AccountName	linda
\$SourceIPAddress	192.168.1.60

21.8.3. Pattern Matching With pm_pattern

Regular expressions are widely used in pattern matching. Unfortunately, using a large number of regular

expression based patterns does not scale well, because these need to be evaluated linearly. The [pm_pattern](#) module implements a more efficient pattern matching than regular expressions used in [Exec](#) directives.

Example 74. Using Regular Expressions With pm_pattern

Syslog Message

```
<38>Nov 22 10:30:12 myhost sshd[8459]: Failed password for invalid user linda from 192.168.1.60  
port 38176 ssh2←
```

With this configuration, the above Syslog message is first parsed with [parse_syslog\(\)](#). This results in a [\\$Message](#) field created in the event record. Then, the [pm_pattern](#) module is used with a pattern XML file to further parse the record.

nxlog.conf

```
1 <Extension _syslog>  
2     Module      xm_syslog  
3 </Extension>  
4  
5 <Input in>  
6     Module      im_udp  
7     Host        0.0.0.0  
8     Port        514  
9     Exec        parse_syslog();  
10 </Input>  
11  
12 <Processor pattern>  
13     Module      pm_pattern  
14     PatternFile /var/lib/nxlog/patterndb.xml  
15 </Processor>  
16  
17 <Output out>  
18     Module      om_null  
19 </Output>  
20  
21 <Route r>  
22     Path        in => pattern => out  
23 </Route>
```

The patterns for the [pm_pattern](#) module instance above are declared in the following [patterndb.xml](#) file.

Pattern Database (patterndb.xml)

```

<?xml version='1.0' encoding='UTF-8'?>
<patterndb>
  <created>2010-01-01 01:02:03</created>
  <version>42</version>
  <!-- First and only pattern group in this file -->
  <group>
    <name>ssh</name>
    <id>42</id>
    <!-- Only try to match this group if $SourceName == "sshd" -->
    <matchfield>
      <name>SourceName</name>
      <type>exact</type>
      <value>sshd</value>
    </matchfield>
    <!-- First and only pattern in this pattern group -->
    <pattern>
      <id>1</id>
      <name>ssh auth failure</name>
      <!-- Do regular expression match on $Message field -->
      <matchfield>
        <name>Message</name>
        <type>regexp</type>
        <value>^Failed (\S+) for(?: invalid user)? (\S+) from (\S+) port \d+ ssh2</value>
      <!-- Set 3 event record fields from captured strings -->
      <capturedfield>
        <name>AuthMethod</name>
        <type>string</type>
      </capturedfield>
      <capturedfield>
        <name>AccountName</name>
        <type>string</type>
      </capturedfield>
      <capturedfield>
        <name>SourceIPAddress</name>
        <type>string</type>
      </capturedfield>
      <!-- Set additional fields if pattern matches -->
      <set>
        <field>
          <name>TaxonomyAction</name>
          <value>Authenticate</value>
          <type>string</type>
        </field>
        <field>
          <name>TaxonomyStatus</name>
          <value>Failure</value>
          <type>string</type>
        </field>
      </set>
    </pattern>
  </group>
</patterndb>

```

Table 51. Fields Added by pm_pattern

Field	Value
\$AuthMethod	password

Field	Value
\$AccountName	linda
\$SourceIPAddress	192.168.1.60
\$TaxonomyAction	Authenticate
\$TaxonomyStatus	Failure

NXLog Manager provides an interface for writing pattern files, and will also test sample events to aid in establishing the correct match patterns. The pattern functions can be accessed from the **PATTERNS** menu in the page header.

Example 75. Creating Patterns With NXLog Manager

The following instructions explain the steps required for creating the above pattern database with NXLog Manager.

1. Open **PATTERNS** > **CREATE GROUP**. Enter a **Name** for the new pattern group, and optionally a **Description**, in the **Properties** section. The name is used to refer to the pattern group later. The name of the above pattern group is **ssh**.
2. Add a match field by clicking [**Add Field**] in the **Match** section. Only messages that match will be further processed by this pattern group. In the above example, there is no reason to attempt any matches if the **\$SourceName** field does not equal **sshd**. The above pattern group uses **Field name=SourceName**, **Match=EXACT**, and **Value=sshd**.

Match			
Field name	Type	Match	Value
SourceName	EXACT	sshd	<input type="button" value="Delete"/>

Add Field

3. Save the new pattern group.
4. Open **PATTERNS** > **CREATE FIELD** to create a new field to be used when creating new patterns. For the above example, the **\$AuthMethod** field must be added because it is not in the default set provided by NXLog Manager. Set **Name=AuthMethod** and **Field Type=STRING**, then click [**Save**].
5. Open **PATTERNS** > **CREATE PATTERN**. In the **Pattern Info** section, enter a **Pattern Name** and optionally a **Pattern Description**. Select the correct **Pattern Group** from the list. In the above example, the **ssh** pattern group is used.
6. In the **Match** section, set match values for the fields to be matched. If a regular expression match with captured subgroups is detected, the interface will provide a **Captured fields** list where target fields can be selected. The above example uses **Field name=Message**, **Match=REGEXP**, and **Value=^Failed (\S+) for(?: invalid user)? (\S+) from (\S+) port \d+ ssh2\$**. The three captured fields are **AuthMethod**, **AccountName**, and **SourceIPAddress**.

Match

Field name	Type	Match	Value
Message	[STRING]	REGEXP	^Failed (\S+) for(?: invalid user)? (\S+) from (\S+) port \d+ ssh2\$
		Captured fields	Type
		AuthMethod	[STRING]
		AccountName	[STRING]
		SourceIPAddress	[IP4ADDR]

Add Field **Update Test Cases**

7. The **Set** section allows fields to be set if the match is successful. Click [**Add Field**] for each field. The above example sets **\$TaxonomyStatus** to **Failure** and **\$TaxonomyAction** to **Authenticate**.

Set

Field name	Type	Value
TaxonomyAction	[STRING]	Authenticate
TaxonomyStatus	[STRING]	Failure

Add Field

8. The **Action** section accepts NXLog language statements like would be specified in an **Exec** directive. Click [**Add action**], type in the statement, and click [**Verify**] to make sure the statement is valid. The above example does not include any NXLog language statements.
9. The final tabbed section allows test messages to be entered to verify that the match works as expected. Click the **[+]** to add a test case. To test the above example, add a **Value** for the **Message** field: **Failed password for invalid user linda from 192.168.1.60 port 38176 ssh2**. Click [**Update Test Cases**] in the **Match** section to automatically fill the captured fields. Verify that the fields are set as expected. Additional test cases can be added to test other events.

Test Case 0 X [+]

Field name	Type	Value
Message	[STRING]	Failed password for invalid user linda from 192.168.1.60 port 38176 ssh2

Add Field

Captured fields

Field name	Type	Value
AuthMethod	[STRING]	password
AccountName	[STRING]	linda
SourceIPAddress	[IP4ADDR]	192.168.1.60

Add Field **Calculate Fields**

10. Save the new pattern. Then click [**Export**] to download the pattern.xml file or use the pattern to configure a managed agent.

See the NXLog Manager User Guide for more information.

21.8.4. Using the Extracted Fields

The previous sections explore ways that the log message can be parsed and new fields added to the event record. Once the required data has been extracted and corresponding fields created, there are various ways to use this new data.

- A field or set of fields can be matched by string or regular expression to trigger [alerts](#), perform [filtering](#), or further [classify](#) the event.
- Fields in the event record can be [renamed](#), [modified](#), or [deleted](#).
- [Event correlation](#) can be used to execute statements or suppress messages based on matching events inside a specified window.
- Some output formats can be used to preserve the full set of fields in the event record (such as [JSON](#) and the NXLog [Binary](#) format).

21.9. Filtering Messages

Message filtering is a process where only some of the received messages are kept. Filtering is possible using regular expressions or other operators using any of the fields.

Use the [drop\(\)](#) procedure in an [Exec](#) directive to conditionally discard messages.

Example 76. Using drop() to Discard Unmatching Messages

In this example, any line that matches neither of the two regular expressions will be discarded with the [drop\(\)](#) procedure. Only lines that match at least one of the regular expressions will be kept.

nxlog.conf

```
1 <Input file>
2   Module im_file
3   File   "/var/log/myapp/*.log"
4   Exec   if not ($raw_event =~ /failed/ or $raw_event =~ /error/) drop();
5 </Input>
6
7 <Output out>
8   Module om_file
9   File   "/var/log/myapp/errors.txt"
10 </Output>
11
12 <Route r>
13   Path   file => out
14 </Route>
```

21.10. Format Conversion

The requirements and possibilities for format conversion are endless. NXLog provides a broad range of functionality for conversion, including the NXLog language and dedicated modules. For special cases a processor or extension module can be crafted.

For converting between CSV formats, see [Complex CSV Format Conversion](#).

Example 77. Converting from BSD to IETF Syslog

This configuration receives log messages in the BSD Syslog format over UDP and forwards the logs in the IETF Syslog format over TCP.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input bsd>
6     Module im_udp
7     Port   514
8     Host   0.0.0.0
9     Exec   parse_syslog_bsd(); to_syslog_ietf();
10 </Input>
11
12 <Output ietf>
13     Module om_tcp
14     Host   1.2.3.4
15     Port   1514
16 </Output>
17
18 <Route bsd_to_ietf>
19     Path   bsd => ietf
20 </Route>
```

21.11. Log Rotation and Retention

NXLog can implement many kinds of log rotation and retention policies in order to prevent over-use of disk space and to organize past logs. These policies can be applied based on file size, time intervals, or even event attributes (such as severity). Log files can be rotated out to custom filenames and then compressed and/or deleted after a specified time period. The configuration is very flexible and custom policies can be easily implemented.

NXLog supports three main approaches to file rotation. In each case, policies should usually be implemented using a [Schedule](#) block.

- Most policies are implemented under the scope of an [om_file](#) module instance, where output files are being written.
- The [im_file](#) module can be configured to rotate log files after they have been fully read.
- Any log file on the system can be rotated under the scope of an [xm_fileop](#) module or any other module. This includes the internal log file (specified by the [LogFile](#) directive).

Example 78. Rotating om_file Log Files

Log files written by an `om_file` module often need to be rotated regularly. This example uses the `om_file file_name()` function and `xm_fileop file_cycle()` procedure to rotate the output file daily, keeping a total of 7 old log files.

nxlog.conf

```

1 <Extension _fileop>
2   Module xm_fileop
3 </Extension>
4
5 <Output out>
6   Module om_file
7   File   '/var/log/out.log'
8 <Schedule>
9   When   @daily
10  <Exec>
11    file_cycle(file_name(), 7);
12    reopen();
13  </Exec>
14 </Schedule>
15 </Output>
```

Example 79. Rotating the Internal Log File

NXLog will write its own logs to a file specified by the `LogFile` directive. It is good practice to set up rotation of this file. This configuration uses the `xm_fileop file_size()` function. The `file_cycle()` procedure rotates the file if it is larger than 5 MB. The file is also rotated weekly. No more than 8 past log files are retained.

nxlog.conf

```

1 define LOGFILE /opt/nxlog/var/log/nxlog/nxlog.log
2 LogFile %LOGFILE%
3
4 <Extension _fileop>
5   Module xm_fileop
6
7   # Check the log file size every hour and rotate if larger than 5 MB
8 <Schedule>
9   Every 1 hour
10  <Exec>
11    if (file_exists('%LOGFILE') and file_size('%LOGFILE%') >= 5M)
12      file_cycle('%LOGFILE%', 8);
13  </Exec>
14 </Schedule>
15
16 # Rotate log file every week on Sunday at midnight
17 <Schedule>
18   When   @weekly
19   Exec   if file_exists('%LOGFILE') file_cycle('%LOGFILE%', 8);
20 </Schedule>
21 </Extension>
```

There are many other ways that rotation and retention can be implemented. See the following sections for more details and examples.

21.11.1. Rotation Policies and Intervals

- The `om_file reopen()` procedure will cause NXLog to reopen the output file specified by the `File` directive.
- The `rotate_to()` procedure can be used to choose a name to rotate the current file to. This procedure will reopen the output file automatically, so there is no need to use the `reopen()` procedure.
- The `file_cycle()` procedure will move the selected file to "`file.1`". If "`file.1`" already exists, it will be moved to "`file.2`", and so on. If an integer is used as a second argument, it specifies the maximum number of previous files to keep.

WARNING

If `file_cycle()` is used on a file that NXLog currently has open under the scope of an `om_file` module instance, the `reopen()` procedure must be used to continue logging to the file specified by the `File` directive. Otherwise, events will continue to be logged to the rotated file ("`file.1`", for example). (This is not necessary if the rotated file is the `LogFile`.)

21.11.1.1. Rotating by File Size

A log file can be rotated according to a pre-defined file size. This policy can be configured with the `om_file file_size()` function or the `xm_fileop file_size()` function.

Example 80. Using the `file_size()` Function

This example uses the `file_size()` function to detect if a file has grown beyond a specified size. If it has, the `file_cycle()` procedure is used to rotate it. The file size is checked hourly with the `When` directive.

nxlog.conf

```
1 <Extension _fileop>
2   Module xm_fileop
3 </Extension>
4
5 <Output out>
6   Module om_file
7   File   '/var/log/out.log'
8   <Schedule>
9     When @hourly
10    <Exec>
11      if file_size(file_name()) >= 1M
12      {
13        file_cycle(file_name());
14        reopen();
15      }
16    </Exec>
17  </Schedule>
18 </Output>
```

21.11.1.2. Using Time-Based Intervals

For time interval based rotation policies NXLog provides two directives for use in `Schedule` blocks.

- The `Every` directive rotates log files according to a specific interval specified in seconds, minutes, days, or weeks.
- The `When` directive provides crontab-style scheduling, including extensions like `@hourly`, `@daily`, and `@weekly`.

Example 81. Using Every and When for Time-Based Rotation

This example shows the use of the [Every](#) and [When](#) directives. The output file is rotated daily using the [rotate_to\(\)](#) function. The name is generated in the **YYYY-MM-DD** format according to the current server time.

nxlog.conf

```

1 <Output out>
2   Module  om_file
3   File    '/var/log/out.log'
4   <Schedule>
5     # This can likewise be used for `@weekly` or `@monthly` time periods.
6     When @daily
7
8     # The following crontab-style is the same as `@daily` above.
9     # When "0 0 * * *"
10
11    # The `Every` directive could also be used in this case.
12    # Every 24 hour
13
14    Exec   rotate_to(file_name() + strftime(now(), '_%Y-%m-%d'));
15  </Schedule>
16 </Output>
```

Example 82. Rotating Into a Nested Directory Structure

In this example, logs for each year and month are stored in separated sub-directories as shown below. The log file is rotated daily.

```
.../logs/YEAR/MONTH/YYYY-MM-DD.log
```

This is accomplished with the *xm_fileop* [dir_make\(\)](#) procedure, the core [strftime\(\)](#) function, and the *om_file* [rotate_to\(\)](#) procedure.

nxlog.conf

```

1 <Extension _fileop>
2   Module  xm_fileop
3 </Extension>
4
5 <Output out>
6   define OUT_DIR /srv/logs
7
8   Module  om_file
9   File    '%OUT_DIR%/out.log'
10  <Schedule>
11    When @daily
12    <Exec>
13      # Create year/month directories if necessary
14      dir_make('%OUT_DIR%/' + strftime(now(), '%Y/%m'));
15
16      # Rotate current file into the correct directory
17      rotate_to('%OUT_DIR%/' + strftime(now(), '%Y/%m/%Y-%m-%d.log'));
18    </Exec>
19  </Schedule>
20 </Output>
```

21.11.1.3. Using Dynamic Filenames

As an alternative to traditional file rotation, output filenames can be set dynamically, based on each log event individually. This is possible because the `om_file` File directive supports expressions.

NOTE

Because dynamic filenames result in events being written to multiple files with semi-arbitrary names, they are not suitable for scenarios where a server or application expects events to be written to a particular `foo.log`. In this case normal rotation should be used instead.

Often one of `now()`, `$EventReceivedTime`, and `$EventTime` are used for dynamic filenames. Consider the following points.

- The `now()` function uses the current server time, not when the event was created or when it was received by NXLog. If logs are delayed, they will be stored according to the time at which the NXLog output module instance processes them. This will not work with `nxlog-processor(8)` (see [Offline Log Processing](#)).
- The `$EventReceivedTime` field timestamp is set by the input module instance when an event is received by NXLog. This will usually be practically the same as using `now()`, except in cases where there are processing delays in the NXLog route (such as when using `Buffering`). This can be used with `nxlog-processor(8)` if the `$EventReceivedTime` field was previously set in the logs.
- The `$EventTime` field is set from a timestamp in the event, so will result in correct value even if the event was delayed before reaching NXLog. Note that some parsing may be required before this field is available (for example, the `parse_syslog()` procedure sets the `xm_syslog EventTime` field). Note also that an incorrect timestamp in an event record can cause the field to be unset or filled incorrectly resulting in data written into the wrong file.

Example 83. Timestamp-Based Dynamic Filenames With `om_file`

This example accepts Syslog formatted messages via UDP. Each message is parsed by the `parse syslog()` procedure. The `EventTime` field is set from the timestamp in the syslog header. This field is then used by the expression in the `File` directive to generate an output filename for the event.

Even if messages received from clients over the network are out of order or delayed, they will still be placed in the appropriate output files according to the timestamps.

`nxlog.conf`

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input in>
6   Module im_udp
7   Port 514
8   Host 0.0.0.0
9   Exec parse_syslog();
10 </Input>
11
12 <Output out>
13   Module om_file
14   File '/var/log/nxlog/out_' + strftime($EventTime, '%Y-%m-%d')
15   Exec to_syslog_ietf();
16 </Output>
```

Dynamic filenames can be based on other fields also.

Example 84. Attribute-Based Dynamic Filenames With `om_file`

In this example, events are grouped by their source hostname.

`nxlog.conf`

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input in>
6   Module im_udp
7   Host 0.0.0.0
8   Port 514
9   Exec parse_syslog();
10 </Input>
11
12 <Output out>
13   Module om_file
14   File '/tmp/logs_by_host/' + $Hostname
15 </Output>
```

21.11.1.4. Rotating Input Files

An *im_file* module instance can be configured to manipulate files after they are fully processed. The *im_file* [OnEOF](#) block can be used for this purpose.

WARNING

When using [OnEOF](#) for rotation, the rotated files must be named (or placed in a directory) such that they will not be detected as new files and re-read by the module instance.

NOTE

If a logging service keeps a log file open for writing, the *xm_exec exec()* procedure should be used to restart the service or otherwise instruct it to re-open the log file.

Example 85. Using im_file OnEOF for Input Files

In this example, files matching `/var/log/app/*.log` are read with an `im_file` module instance. When each file has been fully read, it is rotated. The `GraceTimeout` directive will prevent NXLog from rotating the file until after there have been no events for 10 seconds.

The input files are rotated by adding a timestamp suffix to the filename. For example, an input file named `/var/log/app/errors.log` would be rotated to `/var/log/app/errors.log_20180101T130100`. The new name does not match the wildcard specified by the `File` directive, so the file is not re-read.

`nxlog.conf`

```
1 <Extension _fileop>
2   Module  xm_fileop
3 </Extension>
4
5 <Input app_logs_rotated>
6   Module  im_file
7   File    '/var/log/app/*.log'
8   <OnEOF>
9     <Exec>
10    file_rename(file_name(),
11                file_name() + strftime(now(), '_%Y%m%dT%H%M%S'));
12  </Exec>
13  GraceTimeout 10
14 </OnEOF>
15 </Input>
```

21.11.2. Retention Policies

NXLog can be configured to keep old log files according to a particular retention policy. Functions and procedures for retention are provided by the `xm_fileop` module. Additional actions, such as compressing old log files, can be implemented with the `xm_exec` extension module.

21.11.2.1. Using Simple File Cycling

The `file_cycle()` procedure provides simple numbered rotation and, optionally, retention.

Example 86. Cycling One Year of Logs With file_cycle()

This example demonstrates the use of the `xm_fileop file_cycle()` procedure for keeping a total of 12 log files, one for each month. Log files older than 1 year will be automatically deleted.

This policy creates following log file structure: `/var/log/foo.log` for the current month, `/var/log/foo.log.1` for the previous month, and so on up to the maximum of 12 files.

nxlog.conf

```
1 <Extension _fileop>
2   Module xm_fileop
3 </Extension>
4
5 <Output out>
6   Module om_file
7   File   '/var/logs/foo.log'
8   <Schedule>
9     When @monthly
10    <Exec>
11      file_cycle(file_name(), 12);
12      reopen();
13    </Exec>
14  </Schedule>
15 </Output>
```

Different policies for different events can be implemented in combination with [dynamic filenames](#).

Example 87. Retaining Files According to Severity

This example uses the `$Severity` field (such as `$Severity` set by `parse_syslog()`) to filter events to separate files. Then different retention policies are applied according to severity. Here, one week of debug logs, 2 weeks of informational logs, and 4 weeks of higher severity logs are retained.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _fileop>
6     Module xm_fileop
7 </Extension>
8
9 <Input logs_in>
10    Module im_file
11    File   "/var/log/messages"
12    Exec   parse_syslog();
13 </Input>
14
15 <Output logs_out>
16     define OUT_DIR /opt/nxlog/var/log
17
18     Module om_file
19     File   '%OUT_DIR%/' + $Severity + '.log'
20 <Schedule>
21     When @daily
22     <Exec>
23         file_cycle('%OUT_DIR%/DEBUG.log', 7);
24         file_cycle('%OUT_DIR%/INFO.log', 14);
25         file_cycle('%OUT_DIR%/WARNING.log', 28);
26         file_cycle('%OUT_DIR%/ERROR.log', 28);
27         file_cycle('%OUT_DIR%/CRITICAL.log', 28);
28         reopen();
29     </Exec>
30 </Schedule>
31 </Output>
```

21.11.2.2. Compressing Old Log Files

The `xm_exec` module can be used to compress old log files to reduce disk usage.

Example 88. Using bzip2 With exec_async()

In this example, the file size of the output file is checked hourly with the *om_file file_size()* function. If the size is over the limit, then:

1. a **newfile** module variable is set to the name the current file will be rotated to,
2. the *om_file rotate_to()* procedure renames the current output file to the name set in **newfile**,
3. the module re-opens the original file specified by the **File** directive and continue logging, and
4. the *xm_exec exec_async()* procedure call bzip2 on the rotated-out file (without waiting for the command to complete).

nxlog.conf

```
1 <Extension _exec>
2     Module xm_exec
3 </Extension>
4
5 <Extension _fileop>
6     Module xm_fileop
7 </Extension>
8
9 <Output out>
10    Module om_file
11    File   '/opt/nxlog/var/log/app.log'
12    <Schedule>
13        When @hourly
14        <Exec>
15            if out->file_size() > 15M
16            {
17                set_var('newfile', file_name() + strftime(now(), '_%Y%m%d%H%M%S'));
18                rotate_to(get_var('newfile'));
19                exec_async('/bin/bzip2', get_var('newfile'));
20            }
21        </Exec>
22    </Schedule>
23 </Output>
```

21.11.2.3. Deleting Old Log Files

For retention policies where file deletion is not handled automatically by the *xm_fileop file_cycle()* procedure, the *xm_fileop file_remove()* can be used to delete old files. This procedure can also delete files based on their creation time.

Example 89. Using file_remove() to Delete Old Files

This example uses [file_remove\(\)](#) to remove any files that were created more than 30 days ago.

nxlog.conf

```
1 <Input in>
2   Module im_null
3 </Input>
4
5 <Output logs_out>
6   Module om_file
7   File  '/var/log/' + strftime(now(), '%Y%m%d') + '.log'
8 <Schedule>
9   When @daily
10
11  # Delete logs older than 30 days (24x60x60x30)
12  Exec  file_remove('/var/log/*.log', now() - 2592000);
13 </Schedule>
14 </Output>
```

21.12. Message Classification

Pattern matching is commonly used for message classification. When certain strings are detected in a log message, the message gets tagged with classifiers. Thus it is possible to query or take action based on the classifiers only. There are several ways to classify messages based on patterns.

See also [Extracting Data](#), a closely related topic, for more examples of classification.

21.12.1. Simple Matching on Fields

Often, message classification can be performed during parsing. However, if the required fields have already been parsed or the input module provides structured data, then it is only necessary to match the relevant fields and set classifiers.

Example 90. Classifying a Windows Security EventLog Message

This example classifies Windows Security login failure events with [Event ID 4625](#) (controlled by the "Audit logon events" audit policy setting). If a received event has that ID, it is classified as a failed authentication attempt and the **\$AccountName** field is set to the value of **\$TargetUserName**.

Table 52. Sample Event via im_msvistalog (Excerpt)

Field	Value
\$EventType	AUDIT_FAILURE
\$EventID	4625
\$SourceName	Microsoft-Windows-Security-Auditing
\$Channel	Security
\$Category	Logon
\$TargetUserSid	S-1-0-0
\$TargetUserName	linda
\$TargetDomainName	WINHOST
\$Status	0xc000006d
\$FailureReason	%%2313
\$SubStatus	0xc000006a
\$LogonType	2

nxlog.conf

```

1 <Input in>
2   Module  im_msvistalog
3   <Exec>
4     if ($EventID == 4625) and
5       ($SourceName == 'Microsoft-Windows-Security-Auditing')
6     {
7       $TaxonomyAction = 'Authenticate';
8       $TaxonomyStatus = 'Failure';
9       $AccountName = $TargetUserName;
10    }
11  </Exec>
12 </Input>
```

Table 53. Fields Added to the Event Record

Field	Value
\$TaxonomyAction	Authenticate
\$TaxonomyStatus	Failure
\$AccountName	linda

21.12.2. Regular Expressions via the Exec Directive

The `=~` operator can be used for regular expression matching in an [Exec](#) directive.

Example 91. Regular Expression Message Classification

When the contents of the **\$Message** field match against the regular expression, the **\$AccountName** and **\$AccountID** fields are filled with the appropriate values from the referenced captured sub-strings. Additionally the value **LoginEvent** is stored in the **\$Action** field.

```
1 if $Message =~ /(?x)^pam_unix\(sshd:session\):\ session\ opened\ for\ user\ (\$+)
2           \ by\ \$(uid=(\d+))/ 
3 {
4     $AccountName = $1;
5     $AccountID = integer($2);
6     $Action = 'LoginEvent';
7 }
```

21.12.3. Using pm_pattern

When there are a lot of patterns, writing them all in the configuration file is inefficient. Instead, the **pm_pattern** module can be used.

Example 92. Classifying With pm_pattern

The above pattern matching rule can be defined in the [pm_pattern](#) module's XML format in the following way, which will accomplish the same result.

Pattern Database (patterndb.xml)

```
<pattern>
  <id>42</id>
  <name>ssh_pam_session_opened</name>
  <description>ssh pam session opened</description>
  <matchfield>
    <name>Message</name>
    <type>REGEXP</type>
    <value>
      ^pam_unix\(\sshd:session\): session opened for user (\S+) by \(\uid=(\d+)\\
    </value>
    <capturedfield>
      <name>AccountName</name>
      <type>STRING</type>
    </capturedfield>
    <capturedfield>
      <name>AccountID</name>
      <type>INTEGER</type>
    </capturedfield>
  </matchfield>
  <set>
    <field>
      <name>Action</name>
      <type>STRING</type>
      <value>LoginEvent</value>
    </field>
  </set>
</pattern>
```

nxlog.conf

```
1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input uds>
6   Module      im_uds
7   UDS        /dev/log
8   Exec       parse_syslog_bsd();
9 </Input>
10
11 <Processor pattern>
12   Module      pm_pattern
13   PatternFile /var/lib/nxlog/patterndb.xml
14 </Processor>
15
16 <Output file>
17   Module      om_file
18   File        "/var/log/messages"
19   Exec       to_syslog_bsd();
20 </Output>
21
22 <Route uds_to_file>
23   Path        uds => pattern => file
24 </Route>
```

21.13. Parsing Multi-Line Messages

Multi-line messages such as exception logs and stack traces are quite common in logs. Unfortunately these log messages are often stored in files or forwarded over the network without any encapsulation. In this case, the newline characters in the messages cannot be correctly parsed by simple line-based parsers, which treat every line as a separate event.

Multi-line events may have one or more of:

- a header in the first line (with timestamp and severity field, for example),
- a closing character sequence marking the end, and
- a fixed line count.

Based on this information, NXLog can be configured to reconstruct the original messages, creating a single event for each multi-line message.

21.13.1. xm_multiline

NXLog provides [xm_multiline](#) for multi-line parsing; this dedicated extension module is the recommended way to parse multi-line messages. It supports header lines, footer lines, and fixed line counts. Once configured, the *xm_multiline* module instance can be used as a parser via the input module's [InputType](#) directive.

Example 93. Using the xm_multiline Module

This configuration creates a single event record with the matching [HeaderLine](#) and all successive lines until an [EndLine](#) is received.

```
nxlog.conf
1 <Extension xm_multiline_parser>
2   Module      xm_multiline
3   HeaderLine  "-----"
4   EndLine     "END-----"
5 </Extension>
6
7 <Input in>
8   Module      im_file
9   File        "/var/log/app-multiline.log"
10  InputType   multiline_parser
11 </Input>
```

It is also possible to use [regular expressions](#) with the [HeaderLine](#) and [EndLine](#) directives.

Example 94. Using Regular Expressions With xm_multiline

Here, a new event record is created beginning with each line that matches the regular expression.

nxlog.conf

```
1 <Extension tomcat_parser>
2   Module      xm_multiline
3   HeaderLine  /^"\d{4}[-]\d{2}[-]\d{2} \d{2}:\d{2}:\d{2},\d{3} \s+ [\s+] - .*/
4 </Extension>
5
6 <Input log4j>
7   Module      im_file
8   File        "/var/log/tomcat6/catalina.out"
9   InputType   tomcat_parser
10 </Input>
```

NOTE

With this configuration, a log message is kept in the buffers, and not forwarded, until a new log message is read. This occurs because the [xm_multiline](#) input parser cannot know that a log message is finished until it receives a new [HeaderLine](#). The *xm_multiline* module provides an [EndLine](#) directive for receiving log messages with a closing footer, in which case this does not occur.

21.13.2. Module Variables

It is also possible to parse multi-line messages by using [module variables](#), as shown below. However, it is generally recommended to use the [xm_multiline](#) module instead, because it offers:

- more efficient message processing,
- a more readable configuration,
- correctly incremented module event counters (one increment per multi-line message versus one per line), and
- operation on the message source level rather than the module instance level (each file for a wildcarded [im_file](#) module instance or each TCP connection for an [im_tcp/im_ssl](#) instance).

Example 95. Parsing Multi-Line Messages with Module Variables

This example saves the matching line and successive lines in the `saved` variable. When another matching line is read, an internal log message is generated with the contents of the `saved` variable.

nxlog.conf

```
1 <Input log4j>
2   Module im_file
3   File   "/var/log/tomcat6/catalina.out"
4 <Exec>
5   if $raw_event =~ /(?x)^\\d{4}-\\d{2}\\-\\d{2}\\ \\d{2}\\.\\d{2},\\d{3}\\ \\S+
6       \\ [\\S+]\\ -\\ .*/
7   {
8     if defined(get_var('saved'))
9     {
10       $tmp = $raw_event;
11       $raw_event = get_var('saved');
12       set_var('saved', $tmp);
13       $tmp = undef;
14       log_info($raw_event);
15     }
16     else
17     {
18       set_var('saved', $raw_event);
19       drop();
20     }
21   }
22   else
23   {
24     set_var('saved', get_var('saved') + "\\n" + $raw_event);
25     drop();
26   }
27 </Exec>
28 </Input>
```

NOTE

As with the previous example, a log message is kept in the `saved` variable, and not forwarded, until a new log message is read.

21.14. Rate Limiting

The poor man's tool for rate limiting is the [sleep\(\)](#) procedure.

Example 96. Rate Limiting with sleep()

In the following example `sleep()` is invoked with 500 microseconds. This means that the input module will be able to read at most 2000 messages per second.

nxlog.conf

```
1 <Input in>
2     Module  im_tcp
3     Host    0.0.0.0
4     Port    1514
5     Exec    sleep(500);
6 </Input>
```

This is not very precise because the module can do additional processing which can add some to the total execution time, but it gets fairly close.

WARNING

It is not recommended to use rate limiting on a route that reads logs over UDP.

21.15. Rewriting and Modifying Messages

There are many ways to modify log messages.

21.15.1. Simple Rewrite

A simple rewrite can be done by modifying the `$raw_event` field without parsing the message (with Syslog, for example). Regular expression capturing can be used for this.

Example 97. Simple Rewrite Statement

This statement, when used in an `Exec` directive, will apply the replacement directly to the `$raw_event` field. In this case, a parsing procedure like `parse_syslog()` would not be used.

```
1 if $raw_event =~ /^(aaaa)(replaceME)(.+)/
2     $raw_event = $1 + 'replaceMENT' + $3;
```

Example 98. Converting a Timestamp Format

This example will convert a timestamp field to a different format. Like the previous example, the goal is to modify the `$raw_event` field directly, rather than use other fields and then a procedure like `to_json()` to update `$raw_event`.

The input log format is line-based, with whitespace-separated fields. The first field is a timestamp expressed as seconds since the epoch.

Input Sample

```
1301471167.225121 AChBVvgs1dfHjwhG8 141.143.210.102 5353 224.0.0.251 5353 udp dns - - - S0 - -  
0 D 1 73 0 0 (empty)↔
```

In the output module instance `Exec` directive, the regular expression will match and capture the first field from the line, and remove it. This captured portion is parsed with the `parsedate()` function and used to set the `$EventTime` field. This field is then prepended to the `$raw_event` field to replace the previously removed field.

nxlog.conf

```
1 <Input in>  
2     Module im_file  
3     File   "conn.log"  
4 </Input>  
5  
6 <Output out>  
7     Module om_tcp  
8     Host   192.168.0.1  
9     Port   1514  
10    <Exec>  
11        if $raw_event =~ s/^(\S+)//  
12        {  
13            $EventTime = parsedate($1);  
14            $raw_event = strftime($EventTime, 'YYYY-MM-DDThh:mm:ss.sTZ') +  
15            $raw_event;  
16        }  
17    </Exec>  
18 </Output>
```

Output Sample

```
2011-03-30T00:46:07.225121-07:00 AChBVvgs1dfHjwhG8 141.143.210.102 5353 224.0.0.251 5353 udp  
dns - - - S0 - - 0 D 1 73 0 0 (empty)↔
```

21.15.2. Modifying Fields

A more complex method is to parse the message into fields, modify some fields, and finally reconstruct the message from the fields. This method is much more versatile: it allows rewriting to be done regardless of input and output formats.

Example 99. Rewrite Using Fields

In this example, each Syslog message is received via UDP and parsed with [parse_syslog_bsd\(\)](#). Then, if the **\$Message** field matches the regular expression, the **\$SeverityValue** field is modified. Finally, the [to_syslog_bsd\(\)](#) procedure generates **\$raw_event** from the fields.

nxlog.conf

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input udp>
6   Module im_udp
7   Port 514
8   Host 0.0.0.0
9   Exec parse_syslog_bsd();
10 </Input>
11
12 <Output file>
13   Module om_file
14   File "/var/log/logmsg.txt"
15   <Exec>
16     if $Message =~ /error/ $SeverityValue = syslog_severity_value("error");
17     to_syslog_bsd();
18   </Exec>
19 </Output>
20
21 <Route syslog_to_file>
22   Path udp => file
23 </Route>
```

21.15.3. Renaming and Deleting Fields

In some cases it may be necessary to rename or delete fields.

The simplest way is to use the NXLog language and the [Exec](#) directive.

Example 100. Simple Field Rename

This statement uses [rename_field\(\)](#) to rename the **\$user** field to **\$AccountName**.

```
1 rename_field($user, $AccountName);
```

Example 101. Simple Field Deletion

This statement uses the [delete\(\)](#) procedure to delete the **\$Serial** field.

```
1 delete($Serial);
```

Alternatively, the [xm_rewrite](#) extension module (available in NXLog Enterprise Edition) can be used to rename or delete fields.

Example 102. Using xm_rewrite to Whitelist and Rename Fields

This example uses the `parse_syslog()` procedure to create a set of **Syslog fields** in the event record. It then uses the `Keep` directive to whitelist a set of fields, deleting any field that is not in the list. Finally the `Rename` directive is used to rename the `$EventTime` field to `$Timestamp`. The resulting event record is converted to JSON and sent out via TCP.

nxlog.conf

```

1 <Extension json>
2   Module xm_json
3 </Extension>
4
5 <Extension rewrite>
6   Module xm_rewrite
7   Keep   EventTime, Severity, Hostname, SourceName, Message
8   Rename EventTime, timestamp
9 </Extension>
10
11 <Input in>
12   Module im_file
13   File   '/var/log/messages'
14   Exec   parse_syslog(); rewrite->process();
15 </Input>
16
17 <Output out>
18   Module om_tcp
19   Host   10.0.0.1
20   Port   1514
21   Exec   to_json();
22 </Output>
23
24 <Route r>
25   Path   in => out
26 </Route>
```

Example 103. Using xm_rewrite to Remove Fields

Here is an example Extension block that uses the `Delete` directive to delete all the severity fields. This could be used to prevent severity-based matching (during later processing) on an event source that does not set severity values correctly.

nxlog.conf

```

1 <Extension rewrite>
2   Module xm_rewrite
3   Delete SyslogSeverityValue, SyslogSeverity, SeverityValue, Severity
4 </Extension>
```

21.16. Timestamps

The NXLog core provides functions for parsing timestamps and returning `datetime` values, and functions for generating formatted timestamps from `datetime` values.

21.16.1. Parsing Timestamps

Most timestamps can be parsed with the `parsedate()` function, which will automatically parse any of the supported formats.

Example 104. Parsing a Timestamp With parsedate()

Consider the following line-based input sample. Each record begins with a timestamp followed by a tab.

Input Sample

```
2016-10-11T22:14:15.003Z → machine.example.com → An account failed to log on.←
```

This example configuration uses a regular expression to capture the string up to the first tab. Then the [parsedate\(\)](#) function is used to parse the resulting string and set the **\$EventTime** field to the corresponding **datetime** value. This value can be converted to a timestamp string as required in later processing, either explicitly or as defined by the global [DateFormat](#) directive (see [Formatting Timestamps](#)).

nxlog.conf

```
1 <Input in>
2   Module  im_file
3   File    'in.log'
4   Exec    if $raw_event =~ /^([^\t])\t/ $EventTime = parsedate($1);
5 </Input>
```

TIP

The [parsedate\(\)](#) function is especially useful if the timestamp format varies within the events being processed. A timestamp of any supported format will be parsed. In this example, the timestamp must be at the beginning of the event and followed by a tab character to be matched by the regular expression.

Sometimes a log source will contain a few events with invalid or unexpected formatting. If [parsedate\(\)](#) fails to parse the input string, it will return an [undefined datetime](#) value. This allows the user to configure a fallback timestamp.

Example 105. Using a Fallback Timestamp With parsedate()

This example statement uses a vague regular expression that may in some cases match an invalid string. If [parsedate\(\)](#) fails to parse the timestamp, it will return an [undefined datetime](#) value. In this case, the final line below will set **\$EventTime** to the current time.

```
1 if $raw_event =~ /^(\S+)\s+(\S+)/
2   $EventTime = parsedate($1 + " " + $2);
3
4 # Make sure $EventTime is set
5 if not defined($EventTime) $EventTime = now();
```

TIP

\$EventTime = \$EventReceivedTime could be used instead to set a timestamp according to when the event was received by NXLog.

For parsing more exotic formats, the [strptime\(\)](#) function can be used.

Example 106. Using strftime() to Parse Timestamps

In this input sample, the date and time are two distinct fields delimited by a tab. It also uses a non-standard single digit format instead of fixed width with double digits.

Input Sample

```
2011-5-29 → 0:3:2 GMT → WINDOWSDC → An account failed to log on.←
```

To parse this, a regular expression can be used to capture the timestamp string. This string is then parsed with the [strftime\(\)](#) function.

```
1 if $raw_event =~ ^(\d+-\d+-\d+\t\d+:\d+:\d+ \w+)/
2     $EventTime = strftime($1, '%Y-%m-%d%t%H:%M:%S %Z');
```

21.16.2. Formatting Timestamps

After a timestamp has been parsed to a [datetime](#) value, it will usually need to be converted back to a string at some point before being sent to the output. This can be done automatically by the output configuration.

Example 107. Using the Default Timestamp Formatting

Consider an event record with an **\$EventTime** field (as a [datetime](#) value) and a **\$Message** field. Note that the table below shows the **\$EventTime** value as it is stored internally: as microseconds since the epoch.

Table 54. Sample Event Record

Field	Value
\$EventTime	1493425133541851
\$Message	EXT4-fs (dm-0): mounted filesystem with ordered data mode.

The following output module instance uses the [to_json\(\)](#) procedure without specifying the timestamp format.

nxlog.conf

```
1 <Output out>
2     Module  om_file
3     File    'out.log'
4     Exec    to_json();
5 </Output>
```

The output of the **\$EventTime** field in this case will depend on the [DateFormat](#) directive. The default **DateFormat** is [YYYY-MM-DD hh:mm:ss](#) (local time).

Output Sample

```
{
  "EventTime": "2017-01-02 15:19:22",
  "Message": "EXT4-fs (dm-0): mounted filesystem with ordered data mode."
}
```

NOTE

A different timestamp may be used in some cases, depending on the procedure used to convert the field and the output module. The [to_syslog_bsd\(\)](#) procedure, for example, will use the **\$EventTime** value to generate a RFC 3164 format timestamp regardless of how the [DateFormat](#) directive is set.

Alternatively, the `strftime()` function can be used to explicitly convert a `datetime` value to a string with the required format.

Example 108. Using strftime() to Format Timestamps

Again, consider an event record with an `$EventTime` field (as a `datetime` value) and a `$Message` field. In this example, the `strftime()` function is used with a format string (see the `strftime(3)` manual) to convert `$EventTime` to a string in the local time zone. Then the `to_json()` procedure is used to set the `$raw_event` field.

nxlog.conf

```
1 <Output out>
2   Module  om_file
3   File    'out.log'
4   <Exec>
5     $EventTime = strftime($EventTime, '%Y-%m-%dT%H:%M:%S%z');
6     to_json();
7   </Exec>
8 </Output>
```

Output Sample

```
{  
  "EventTime": "2017-04-29T02:18:53+0200",  
  "Message": "EXT4-fs (dm-0): mounted filesystem with ordered data mode."  
}
```

NXLog Enterprise Edition supports a few additional format strings for formats that the stock C `strftime()` does not offer, including formats with fractional seconds and in UTC time. See the Reference Manual `strftime()` documentation for the list.

Example 109. Using strftime() Special Formats in NXLog Enterprise Edition

The following statement will convert `$EventTime` to a timestamp format with fractional seconds and in UTC (regardless of the current time zone).

```
1 $EventTime = strftime($EventTime, 'YYYY-MM-DDThh:mm:ss.sUTC');
```

The resulting timestamp string in this case would be `2017-04-29T00:18:53.541851Z`.

Chapter 22. Forwarding and Storing Logs

This chapter discusses the configuration of NXLog outputs, including:

- converting log messages to various [formats](#),
- forwarding logs over the [network](#),
- writing logs to [files and sockets](#),
- storing logs in [databases](#), and
- sending logs to an [executable](#).

22.1. Generating Various Formats

The data format used in an outgoing log message must be considered in addition to the transport protocol. If the message cannot be parsed by the receiver, it may be discarded or improperly processed. See also [Parsing Various Formats](#).

Syslog

There are two Syslog formats, the older BSD Syslog (RFC 3164) and the newer IETF Syslog (RFC 5424). The transport protocol in Syslog can be UDP, TCP, or SSL. The [xm_syslog](#) module provides procedures for generating Syslog messages. For more information, see [Generating Syslog](#).

Example 110. Generating Syslog and Sending via TCP

This configuration uses the [to_syslog_ietf\(\)](#) procedure to convert the corresponding fields in the event record to a Syslog message in IETF format. The result is forwarded via TCP by the [om_tcp](#) module.

```
nxlog.conf
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Output out>
6   Module om_tcp
7   Host 192.168.1.1
8   Port 1514
9   Exec to_syslog_ietf();
10 </Output>
```

Syslog Snare

The Snare agent format is a special format on top of BSD Syslog which is used and understood by several tools and log analyzer frontends. This format is most useful when forwarding Windows EventLog data in conjunction with [im_mseventlog](#) and/or [im_msvisalog](#). The [to_syslog_snare\(\)](#) procedure can construct Syslog Snare formatted messages. For more information, see [Generating Snare](#).

Example 111. Generating Syslog Snare and Sending via UDP

In this example, the [to_syslog_snare\(\)](#) procedure converts the corresponding fields in the event record to Snare format. The messages are then forwarded via UDP by the [om_udp](#) module.

nxlog.conf

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Output out>
6   Module om_udp
7   Host 192.168.1.1
8   Port 514
9   Exec  to_syslog_snare();
10 </Output>
```

NXLog Binary format

The [Binary](#) format is only understood by NXLog. All the fields are preserved when the data is sent in this format, so there is no need to parse it again. The output module instance must contain [OutputType Binary](#). The receiver NXLog module instance can be set to [InputType Binary](#).

Graylog Extended Log Format (GELF)

The [xm_gelf](#) module can be used to generate GELF output.

Example 112. Generating GELF Output

With this configuration, NXLog will send the fields in the event record via UDP in GELF format.

nxlog.conf

```

1 <Extension _gelf>
2   Module xm_gelf
3 </Extension>
4
5 <Output out>
6   Module om_udp
7   Host 127.0.0.1
8   Port 12201
9   OutputType GELF_UDP
10 </Output>
```

22.2. Forwarding over the Network

The following network protocols can be used. There is a trade-off between speed, reliability, compatibility, and security.

UDP

To send logs in UDP datagrams, use the [om_udp](#) module.

WARNING

UDP packets can be dropped by the operating system because the protocol does not guarantee reliable message delivery. It is recommended to use TCP or TLS/SSL instead if message loss is a concern.

Example 113. Using the om_udp Module

This example sends the contents of the **\$raw_event** field to the specified host via UDP.

nxlog.conf

```
1 <Output out>
2   Module  om_udp
3   Host    192.168.1.1
4   Port    514
5 </Output>
```

TCP

To send logs over TCP, use the [om_tcp](#) module.

Example 114. Using the om_tcp Module

In this example, the contents of the **\$raw_event** field are sent to the specified host via TCP.

nxlog.conf

```
1 <Output out>
2   Module  om_tcp
3   Host    192.168.1.1
4   Port    1514
5 </Output>
```

SSL/TLS

To send logs over a trusted secure SSL connection, use the [om_ssl](#) module.

Example 115. Using the om_ssl Module

This example provides nearly identical behavior to the TCP example above, but in this case SSL is used to securely transmit the data.

nxlog.conf

```
1 <Output out>
2   Module  om_ssl
3   Host    192.168.1.1
4   Port    6514
5   CAFile  %CERTDIR%/ca.pem
6   CertFile %CERTDIR%/client-cert.pem
7   CertKeyFile %CERTDIR%/client-key.pem
8 </Output>
```

HTTP(S)

To send logs over a HTTP or HTTPS connection, use the [om_http](#) and [im_http](#) modules (available only in NXLog Enterprise Edition).

Example 116. Using the `om_http` Module

With this configuration, NXLog will send log messages via HTTP, using a POST request for each log message.

`nxlog.conf`

```
1 <Output out>
2   Module  om_http
3   URL     http://server:8080/
4 </Output>
```

22.3. Sending to Files and Sockets

Files

To store logs in local files, use the `om_file` module. See also [Writing Syslog to File](#).

Example 117. Using the `om_file` Module

This configuration writes log messages to the specified file. No additional processing is performed by the output module instance.

`nxlog.conf`

```
1 <Output out>
2   Module  om_file
3   File    "/var/log/out.log"
4 </Output>
```

Unix Domain Socket

To send logs to a Unix domain socket, use the `om_uds` module. See also [Sending Syslog to the Local Syslog Daemon via /dev/log](#).

Example 118. Using the `om_uds` Module

With this configuration, log messages are written to the specified socket without any additional processing.

`nxlog.conf`

```
1 <Output out>
2   Module  om_uds
3   UDS    /dev/log
4 </Output>
```

22.4. Storing in Databases

The `om_dbi` and `om_odbc` modules can be used to store logs in databases. The `om_dbi` module can be used on POSIX systems where libdbi is available. The `om_odbc` module, available in NXLog Enterprise Edition, can be used with ODBC compatible databases on Windows, Linux, and Unix.

Example 119. Using the om_dbi Module

This configuration uses libdbi and the `pgsql` driver to insert events into the specified database. The SQL statement references fields in the event record to be added to the database.

nxlog.conf

```

1 <Output out>
2   Module    om_dbi
3   SQL      INSERT INTO log (facility, severity, hostname, timestamp, application, \
4                           message) \
5         VALUES ($SyslogFacility, $SyslogSeverity, $Hostname, '$EventTime', \
6                  $SourceName, $Message)
7   Driver    pgsql
8   Option    host 127.0.0.1
9   Option    username dbuser
10  Option   password secret
11  Option   dbname logdb
12 </Output>
```

Example 120. Using the om_odbc Module

This example inserts events into the database specified by the ODBC connection string. In this case, the `sql_exec()` and `sql_fetch()` functions are used to interact with the database.

nxlog.conf

```

1 <Output out>
2   Module        om_odbc
3   ConnectionString DSN=mysql_ds;username=mysql;password=mysql;database=logdb;
4   <Exec>
5     if ( sql_exec("INSERT INTO log (facility, severity, hostname, timestamp,
6                     application, message) VALUES (?, ?, ?, ?, ?, ?)",
7                     1, 2, "host", now(), "app", $raw_event) == TRUE )
8     {
9       if ( sql_fetch("SELECT max(id) as id from log") == TRUE )
10       {
11         log_info("ID: " + $id);
12         if ( sql_fetch("SELECT message from log WHERE id=?", $id) == TRUE )
13           log_info($message);
14       }
15     }
16   </Exec>
17 </Output>
```

22.5. Sending to Executables

Using the `om_exec` module, all messages can be piped to an external program or script which will run until the module (or NXLog) is stopped.

Example 121. Using the om_exec Module

This configuration executes the specified command and writes log messages to its standard input.

nxlog.conf

```
1 <Output out>
2   Module  om_exec
3   Command /usr/bin/someprog
4   Arg     -
5 </Output>
```

Chapter 23. Centralized Log Collection

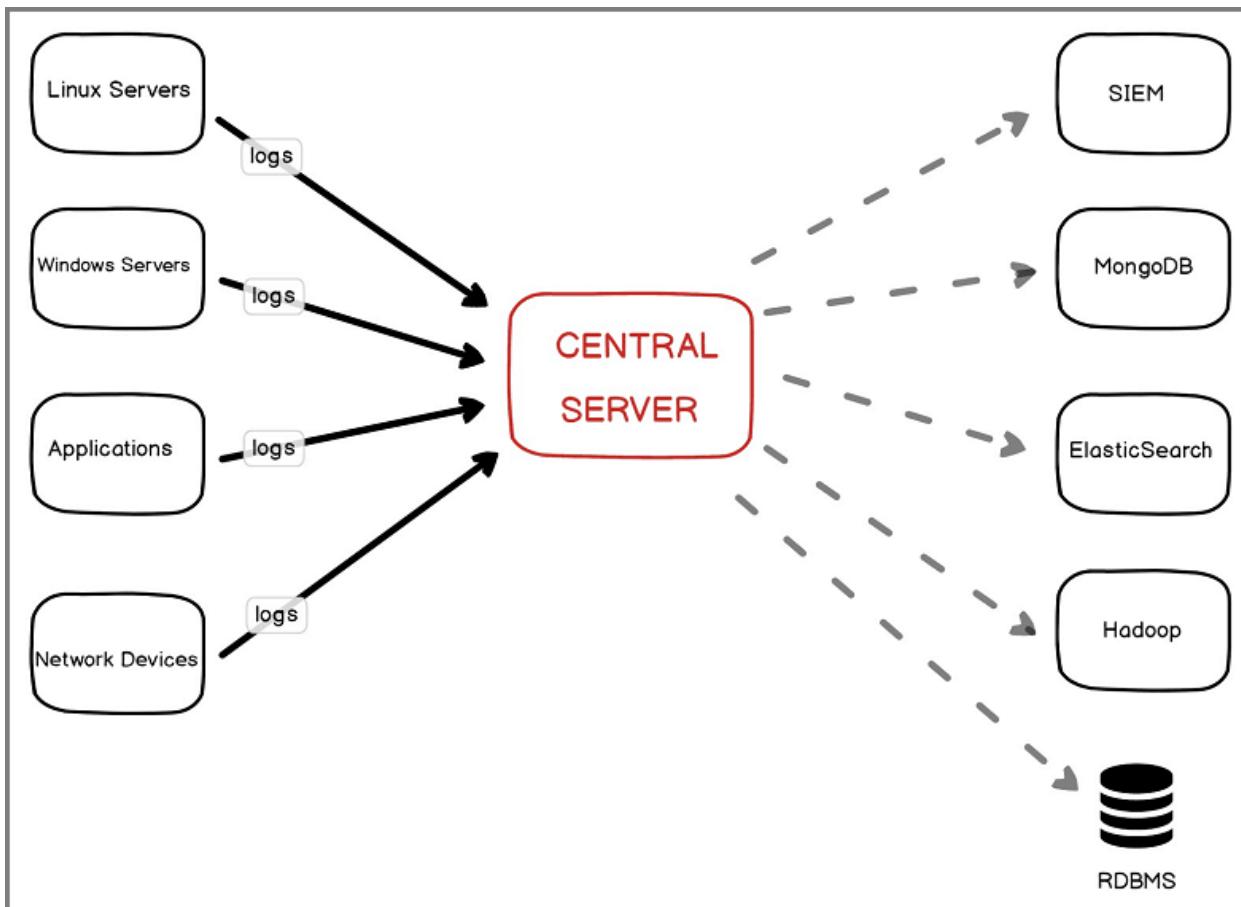
Centralized log collection, log aggregation, or log centralization is the process of sending event log data to a dedicated server or service for storage and optionally search and analytics. Storing logs on a centralized system provides several benefits versus storing the data locally.

- Event data can be accessed even if the originating server is offline, compromised, or decommissioned.
- Data can be analyzed and correlated across more than one system.
- It is more difficult for malicious actors to remove evidence from logs that have already been forwarded.
- Incident investigation and auditing is easier, as all event data is collected in one location.
- Scalable, high-availability, and redundancy solutions are easier to implement and maintain because they can be implemented at the point of the collection server.
- Compliance with internal and external standards for log data retention can be managed at a single point.

23.1. Architecture

The following diagram shows a simple centralized log architecture. The single central server collects logs from servers, applications, and network devices. After collection, the logs can be forwarded as required for further analysis or storage.

This chapter is concerned with the left half of the diagram: collecting logs from clients.



In practice, network topology and other requirements may dictate that additional servers such as relays be added for log handling. For those cases, other functionality may be necessary than what is covered here (such as [buffering](#)).

23.2. Collection Modes

In the context of clients generating logs, NXLog supports both "agent-based" and "agent-less" log collection, and it is possible to set up a system to work in mixed mode. In brief, these modes differ as follows (see the [Log Processing Modes](#) section for more details).

Agent-based log collection requires that an NXLog agent be installed on the client. With a local agent, collection is much more flexible, providing features such as [filtering](#) on the source system to send only the required data, [format conversion](#), compression, encryption, and [delivery reliability](#), among others. It is generally recommended that NXLog be deployed as an agent wherever possible.

Example 122. Transporting Batch-Compressed Logs in Agent-Based Mode

With agent-based log collection, NXLog agents are installed on both the client and the central server. Here, the [im_batchcompress](#) and [om_batchcompress](#) modules are used to transport logs both compressed and encrypted. These modules preserve all the fields in the event record.

nxlog.conf (Client)

```
1 <Output batch>
2   Module    om_batchcompress
3   Host      192.168.56.101
4   Port      2514
5   UseSSL   TRUE
6   CAFile    /opt/openssl_rootca/rootCA.pem
7   CertFile  /opt/openssl_server/server.crt
8   CertKeyFile /opt/openssl_server/server.key
9 </Output>
```

nxlog.conf (Log Server)

```
1 <Input batch>
2   Module    im_batchcompress
3   ListenAddr 0.0.0.0
4   Port      2514
5   CAFile    /opt/openssl_rootca/rootCA.pem
6   CertFile  /opt/openssl_server/central.crt
7   CertKeyFile /opt/openssl_server/central.key
8 </Input>
```

In agent-less mode, there is no NXLog agent installed on the client. Instead, the client forwards events to the central server in a native format. On the central server, NXLog accepts and parses the logs received. Often there is limited control over the log format used, and it may not be possible to implement encryption, compression, delivery reliability, or other features.

Example 123. Collecting UDP Syslog Logs in Agent-Less Mode

With agent-less collection, NXLog is installed on the central server but not on the client. Clients can be configured to send UDP Syslog messages to the central server using their native logging functionality. The `im_udp` module below could be replaced `im_tcp` or `im_ssl` according to what protocol is supported by the clients.

WARNING

UDP transport does not provide any guarantee of delivery. Network congestion or other issues may result in lost log data.

nxlog.conf (Log Server)

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input input_udp>
6   Module im_udp
7   Host   0.0.0.0
8   Port   514
9   Exec   parse_syslog();
10 </Input>
```

It is common for logs to be collected using a mix of different modes among the various clients, network devices, relays, and log servers in a network. For example, an NXLog relay may be configured to collect logs from both agents and agent-less sources and do filtering and processing before forwarding the data to a central server.

23.3. Requirements

Various logging requirements may dictate particular details about the chosen logging architecture. The following are important things to consider when deciding how to set up centralized log collection. In some cases, these requirements can only be met when using agent-based collection.

Reliability

UDP does not guarantee message delivery, and should be avoided if log data loss is not acceptable. Instead, TCP (and therefore, TLS) offers guaranteed packet delivery. In addition, with agent-based collection, NXLog can provide application-level guaranteed delivery. See [Reliable Network Delivery](#) for more information.

Structured data

Correlating data across multiple log sources requires parsing event data into a common set of fields. [Event fields](#) are a core part of NXLog processing, and an NXLog agent can be configured to parse events at any point along the log path. Often, parsing is done as early as possible (at the source, for agent-based collection) to simplify later classifying as reduce processing load on log servers as logs are transported. See [Parsing Various Formats](#) and [Message Classification](#).

Encryption

To maintain confidentiality of log data, TLS can be used during transport.

Compression

If bandwidth is a concern, log data compression may be desirable. Event data is normally quite compressible, allowing bandwidth requirements to be reduced significantly. The `im_batchcompress` and `om_batchcompress` modules provide a batched, compressed transport for log data between NXLog agents.

Storage format

Normally, data should be converted to and stored in a unified format in case of heterogeneous logs sources.

23.4. Data Formats

When using agent-based collection, it is often desirable to convert the data prior to transfer. In this case, structured data is often sent using one of these formats.

Batch compression modules

The [im_batchcompress](#) and [om_batchcompress](#) modules can be used to send logs in compressed and optionally encrypted batches. All fields in the event record are preserved.

NXLog binary format

NXLog has its own binary format (see [Binary InputType](#) and [Binary OutputType](#)) that retains all the fields of an event and can be used to send logs via TCP, UDP, or TLS (or with other stream-oriented modules).

JSON

JSON is easy to generate and parse and has become a de-facto standard for logging as well. It has some limitations such as the missing datetime format. See the [JSON](#) section.

Agent-less collection is restricted to formats supported by the clients. The following are a few common formats, but many more are supported. See also the [OS Support](#) chapters.

Syslog

Using Syslog has become a common practice and many SIEM vendors and products support (and may even require) Syslog. See the [Syslog](#) chapter for more details. Syslog contains free form message data that typically needs to be parsed to extract more information for further analysis. Syslog often uses UDP, TCP, or TLS as transport.

Snare

The [Snare](#) format is commonly used to transport Windows EventLog, with or without Syslog headers.

Windows Event Forwarding (WEF)

Windows EventLog can be forwarded over HTTPS with Windows Event Forwarding. See the [Windows EventLog](#) chapter.

Chapter 24. Encrypted Transfer

In order to protect log data from being modified or viewed by an attacker during transit, NXLog provides SSL/TLS data encryption support in many input and output modules. Benefits of using SSL/TLS encrypted log transfer include:

- strong authentication,
- message integrity (assures that the logs are not changed), and
- message confidentiality (assures that the logs cannot be viewed by an unauthorized party).

WARNING

It is important that certificates be renewed before expiration. The NXLog Manager [dashboard](#) can be configured with a "Certificate summary" which lists soon-to-expire certificates in a separate group.

24.1. SSL/TLS Encryption in NXLog

The SSL/TLS protocol encrypts the log data on the client side and then decrypts it on the server side. It's recommended to use at least 2048-bits keys.

There are several modules in NXLog Enterprise Edition that support SSL/TLS encryption:

- [im_ssl](#) and [om_ssl](#) support secured TCP connections,
- [im_http](#) and [om_http](#) support secured HTTP connections, and
- [im_batchcompress](#) and [om_batchcompress](#) support encryption of compressed log batches transferred between NXLog instances.

When using the SSL/TLS, there are two ways to handle authentication.

- With **mutual authentication**, both client and log server agents are authenticated, and certificates/keys must be deployed for both agents. This is the most secure and prevents log collection if the client's certificate is untrusted or expires.
- With **server-side authentication only**, only the log server is authenticated. A certificate/key must be deployed for the server only. On the log server, the `im_ssl AllowUntrusted` directive (or corresponding directive for [im_http](#) or [im_batchcompress](#)) must be set to TRUE. The client is prevented from sending logs to an untrusted server but the server accepts logs from untrusted clients.

Example 124. Client/Server Encrypted Transfer

With the following configurations, a client reads logs from all log files under the `/var/log` directory, parses the events with `parse_syslog()`, converts to JSON with `to_json()`, and forwards them over a secure connection to the central server.

These configurations use mutual authentication: both agents are authenticated and certificates must be created for both agents.

nxlog.conf (Client)

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module      xm_json
7 </Extension>
8
9 <Input messages>
10  Module     im_file
11    File      "/var/log/*"
12    Exec      parse_syslog();
13 </Input>
14
15 <Output central_ssl>
16   Module     om_ssl
17   Host       192.168.56.103
18   Port       516
19   CAFile    /opt/ssl/rootCA.pem
20   CertFile  /opt/ssl/client.crt
21   CertKeyFile /opt/ssl/client.key
22   KeyPass   password
23   Exec      to_json();
24 </Output>
```

The server receives the logs on port 516 and writes them to `/var/log/logmsg.log`.

nxlog.conf (Central Server)

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input input_ssl>
6   Module     im_ssl
7   Host       0.0.0.0
8   Port       516
9   CAFile    /opt/ssl/rootCA.pem
10  CertFile  /opt/ssl/central.crt
11  CertKeyFile /opt/ssl/central.key
12  KeyPass   password
13 </Input>
14
15 <Output fileout>
16   Module     om_file
17   File      "/var/log/logmsg.log"
18 </Output>
```

24.2. OpenSSL Certificate Creation

[NXLog Manager](#) provides various features for creating, deploying, and managing SSL/TLS certificates, and is especially helpful when managing many NXLog agents across an organization. This section, however, provides steps for creating self-signed certificates with OpenSSL, a Linux-based SSL/TLS cryptography toolkit.

1. Generate the private root key for your Certification Authority (CA).

```
$ openssl genrsa -out rootCA.key 2048
```

2. Self-sign the key and create a CA certificate.

```
$ openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 730 -out rootCA.pem
```

3. Create a certificate for each server.

- a. Generate a private key for the server.

```
$ openssl genrsa -out server.key 2048
```

- b. Generate the certificate signing request for the CA. When prompted for the **Common Name**, enter the server's name or IP address.

```
$ openssl req -new -key server.key -out server.csr
```

- c. Sign the request.

```
$ openssl x509 -req -in server.csr -CA rootCA.pem -CAkey rootCA.key \
-CACreateSerial -out server.crt -days 500 -sha256
```

Chapter 25. Reliable Message Delivery

Sometimes regulatory compliance or other requirements mandate that the logging infrastructure function in an ultra-reliable manner. NXLog Enterprise Edition can be configured to guarantee that:

- log data is safe even [in case of a crash](#),
- no messages are lost due to intermittent [network issues](#), and
- there is [no message duplication](#).

25.1. Crash-Safe Operation

A host or NXLog crash can happen for various reasons, including power failures without a UPS, kernel panics, and software bugs. To protect against data loss in these situations, the following techniques are implemented in NXLog Enterprise Edition.

- Log messages are buffered in various places in NXLog, and buffered messages can be lost in the case of a crash. Persistent module message queues can be enabled so that these messages are stored on disk instead of in memory. Each log message is removed from the queue only after successful delivery. See the [PersistLogqueue](#) and [SyncLogqueue](#) global configuration directives, and the [PersistLogqueue](#) and [SyncLogqueue](#) module directives.

WARNING

Log message removal from queues in processor modules happens before delivery. This can result in potential data loss. Do not use processor modules when high reliability operation is required.

- Input positions (for [im_file](#) and other modules) are saved in the cache file, and by default this file is only saved to disk on shutdown. In case of a crash some events may be duplicated or lost depending on the value of the [ReadFromLast](#) directive. This data can be periodically flushed and synced to disk using the [CacheFlushInterval](#) and [CacheSync](#) directives.

Example 125. Configuration for Crash-Safe Operation

In this example, the log queues are synced to disk after each successful delivery. The cache file containing the current event ID is also flushed and synced to disk after each event is read from the database. Note that these reliability features, when enabled, significantly reduce the processing speed.

nxlog.conf

```
1 PersistLogqueue TRUE
2 SyncLogqueue TRUE
3 CacheFlushInterval always
4 CacheSync TRUE
5
6 <Input in>
7   Module im_file
8   File   'input.log'
9 </Input>
10
11 <Output out>
12   Module om_tcp
13   Host   10.0.0.1
14   Port   1514
15 </Output>
```

25.2. Reliable Network Delivery

The TCP protocol provides guaranteed packet delivery via packet level acknowledgment. Unfortunately, if the receiver closes the TCP connection prematurely while messages are being transmitted, unsent data stored in the socket buffers will be lost since this is handled by the operating system instead of the application (NXLog). This can result in message loss and affects [im_tcp](#), [om_tcp](#), [im_ssl](#), and [om_ssl](#).

The solution to this unreliability in the TCP protocol is application-level acknowledgment. NXLog provides two pairs of modules for this purpose.

- NXLog can use the HTTP/HTTPS protocol to provide guaranteed message delivery over the network, optionally with TLS/SSL. The client ([om_http](#)) sends the event in a HTTP POST request. The server ([im_http](#), only available in NXLog Enterprise Edition) responds with a status code indicating successful message reception.

Example 126. HTTPS Log Transfer

In the following configuration example, a client reads logs from a file and transmits the logs over an SSL-secured HTTP connection.

nxlog.conf (Client/Sending)

```
1 <Input in>
2   Module      im_file
3   File        'input.log'
4 </Input>
5
6 <Output out>
7   Module      om_http
8   URL         https://10.0.0.1:8080/
9   HTTPSCertFile %CERTDIR%/client-cert.pem
10  HTTPSCertKeyFile %CERTDIR%/client-key.pem
11  HTTPSCAFile  %CERTDIR%/ca.pem
12 </Output>
```

The server side accepts the HTTPS connections and stores the received messages in a file. The contents of [input.log](#) will be replicated in [output.log](#).

nxlog.conf (Server/Receiving)

```
1 <Input in>
2   Module      im_http
3   ListenAddr  0.0.0.0
4   Port        8080
5   HTTPSCertFile %CERTDIR%/server-cert.pem
6   HTTPSCertKeyFile %CERTDIR%/server-key.pem
7   HTTPSCAFile  %CERTDIR%/ca.pem
8 </Input>
9
10 <Output out>
11  Module      om_file
12  File        'output.log'
13 </Output>
```

- The [om_batchcompress](#) and [im_batchcompress](#) modules, available in NXLog Enterprise Edition, also provide acknowledgment as part of the batchcompress protocol.

Example 127. Batched Log Transfer

With the following configuration, a client reads logs from a file and transmits the logs in compressed batches to a remote NXLog agent.

nxlog.conf (Client/Sending)

```

1 <Input in>
2   Module      im_file
3   File        'input.log'
4 </Input>
5
6 <Output out>
7   Module      om_batchcompress
8   Host        10.0.0.1
9   UseSSL     true
10  CertFile   %CERTDIR%/client-cert.pem
11  CertKeyFile %CERTDIR%/client-key.pem
12  CAFile     %CERTDIR%/ca.pem
13 </Output>
```

The remote NXLog agent receives and decompresses the received message batches and stores the individual messages in a file. The contents of `input.log` will be replicated in `output.log`.

nxlog.conf (Server/Receiving)

```

1 <Input in>
2   Module      im_batchcompress
3   ListenAddr 0.0.0.0
4   CertFile   %CERTDIR%/server-cert.pem
5   CertKeyFile %CERTDIR%/server-key.pem
6   CAFile     %CERTDIR%/ca.pem
7 </Input>
8
9 <Output out>
10  Module     om_file
11  File       'output.log'
12 </Output>
```

25.3. Protection Against Duplication

If the contents of the cache file (which stores the event position) are lost, the module can either read everything from the beginning or risk losing some messages. In the former case, messages may be duplicated. When using persistent queues, after the output module delivers the message it removes the message from the queue. If the crash occurs just before the removal, the message will be sent again (a duplicate) after the restart.

In some cases it may be very important that a log message not be duplicated. For example, a duplicated message may trigger the same alarm a second time or cause an extra entry in a financial transaction log. NXLog Enterprise Edition can be configured to prevent duplicate messages from occurring.

The best way to prevent duplicated messages is with a serial number, as it is only possible to detect duplicates at the receiver. The receiver can keep track of what has been received by storing the serial number of the last message. If a message is received with the same or a lower serial number from the same source, the message is simply discarded.

In NXLog Enterprise Edition, duplication prevention works as follows.

- Each module which receives a message directly from an input source or from another module in the route assigns a field named `$__SERIAL__$` with a monotonically increasing serial number. The serial number is

taken from a global generator and is increased after each fetch so that two messages received at two modules simultaneously will not have the same serial number. The serial number is initialized to the seconds elapsed since epoch when NXLog is started. This way it can provide 1,000,000 serial numbers per second without problems in case it is stopped and restarted. Otherwise the value would need to be saved and synced to disk as well after each serial number fetch and again this would result in a huge performance hit. When a module receives a message it checks the value of the field named `$_SERIAL$` against the last saved value.

- The [im_http](#) module keeps the value of the last `$_SERIAL$` for each client. It is only possible to know and identify the client ([om_http](#) sender) in HTTPS mode. The Common Name (CN) in the certificate subject is used and is assumed to uniquely identify the client.

NOTE

The remote IP and port number cannot be used to identify the remote sender because the remote port is assigned dynamically and changes for every connection. Thus if a client sends a message, disconnects, reconnects, and then sends the same message again, it is impossible to know if this is the same client or another. For this reason it is not possible to protect against message duplication with plain TCP or HTTP when multiple clients connect from the same IP. The [im_ssl](#) and [im_batchcompress](#) modules do not have the certificate subject extraction implemented at this time.

- All other non-network modules use the value of `SourceModuleName` which is automatically set to the name of the module instance generating the log message. This value is assumed to uniquely identify the source. The value of `SourceModuleName` is not overwritten if it already exists. Note that this may present problems in some complex setups.
- The algorithm is implemented in one procedure call named [duplicate_guard\(\)](#), which can be used in modules to prevent message duplication. The [dropped\(\)](#) function can be then used to test whether the current log message has been dropped.

Example 128. Disallowing Duplicated Messages

The following client and server configuration examples extend the earlier [HTTPS example](#) to provide an ultra-reliable operation where messages cannot be lost locally due to a crash, lost over the network, or duplicated.

nxlog.conf (Client/Sending)

```
1 PersistLogqueue TRUE
2 SyncLogqueue TRUE
3 CacheFlushInterval always
4 CacheSync TRUE
5
6 <Input in>
7   Module      im_file
8   File        'input.log'
9 </Input>
10
11 <Output out>
12   Module      om_http
13   URL         https://10.0.0.1:8080/
14   HTTPSCertFile %CERTDIR%/client-cert.pem
15   HTTPSCertKeyFile %CERTDIR%/client-key.pem
16   HTTPSCAFile  %CERTDIR%/ca.pem
17   Exec        duplicate_guard();
18 </Output>
```

The server side accepts the HTTPS connections and stores the received messages in a file. The contents of `input.log` will be replicated in `output.log`

nxlog.conf (Server/Receiving)

```
1 PersistLogqueue TRUE
2 SyncLogqueue TRUE
3 CacheFlushInterval always
4 CacheSync TRUE
5
6 <Input in>
7   Module      im_http
8   ListenAddr  0.0.0.0
9   Port        8080
10  HTTPSCertFile %CERTDIR%/server-cert.pem
11  HTTPSCertKeyFile %CERTDIR%/server-key.pem
12  HTTPSCAFile  %CERTDIR%/ca.pem
13  Exec        duplicate_guard();
14 </Input>
15
16 <Output out>
17   Module      om_file
18   File        'output.log'
19   Exec        duplicate_guard();
20 </Output>
```

OS Support

Chapter 26. IBM AIX

NXLog can collect various types of system logs on the AIX platform. See the [Supported Platforms](#) section for the list of supported IBM AIX versions.

Local Syslog

Messages written to `/dev/log` can be collected with the `im_uds` module. Events written to file in Syslog format can be collected with `im_file`. In both cases, the `xm_syslog` module can be used to parse the events. See the [Syslog](#) section for more information.

Example 129. Reading Syslog Messages From File

This example reads Syslog messages from `/var/log/messages` and parses them with the `parse_syslog()` procedure.

`nxlog.conf`

```
1 <Extension _syslog>
2   Module  xm_syslog
3 </Extension>
4
5 <Input in>
6   Module  im_file
7   File    "/var/log/messages"
8   Exec    parse_syslog();
9 </Input>
```

AIX Audit

The `im_aixaudit` module natively collects logs generated by the AIX Audit system, without depending on `auditstream` or any other process.

Example 130. Collecting AIX Audit Logs

This example reads AIX audit logs from the `/dev/audit` device file.

`nxlog.conf`

```
1 <Input in>
2   Module      im_aixaudit
3   DeviceFile  /dev/audit
4 </Input>
```

Process Accounting

The `im_acct` module can be used to gather details about who runs what processes.

Example 131. Reading Process Accounting Logs

This configuration turns on process accounting (using `/tmp/nxlog.acct` as the log file) and watches for messages.

`nxlog.conf`

```
1 <Input acct>
2   Module  im_acct
3   AcctOn  TRUE
4   File    "/tmp/nxlog.acct"
5 </Input>
```

File Integrity Monitoring

The [im_fim](#) module can be used to monitor files for changes.

Example 132. Monitoring File Integrity

This example monitors files in the `/etc` and `/srv` directories, generating events when files are modified or deleted. Files ending in `.bak` are excluded from the watch list.

nxlog.conf

```
1 <Input fim>
2   Module      im_fim
3   File        "/etc/*"
4   File        "/srv/*"
5   Exclude     ".*.bak"
6   Digest      sha1
7   ScanInterval 3600
8 </Input>
```

Log Files

The [im_file](#) module can be used to collect events from log files.

Example 133. Reading From Log Files

This configuration reads messages from the `/opt/test/input.log` file. No parsing is performed; each line is available in the `$raw_event` field.

nxlog.conf

```
1 <Input in>
2   Module  im_file
3   File    "/opt/test/input.log"
4 </Input>
```

Custom Programs

The [im_exec](#) module allows log data to be collected from custom external programs.

Example 134. Using an External Command

This example uses the `tail` command to read from a file.

NOTE

The [im_file](#) module should be used to read log messages from files. This example only demonstrates the use of the [im_exec](#) module.

nxlog.conf

```
1 <Input exec>
2   Module  im_exec
3   Command /usr/bin/tail
4   Arg     -f
5   Arg     /var/adm/ras/errlog
6 </Input>
```

Chapter 27. FreeBSD and OpenBSD

NXLog can collect various types of system logs on FreeBSD and OpenBSD platforms. See the [Supported Platforms](#) section for the list of supported FreeBSD and OpenBSD versions.

Kernel

Logs from the kernel can be collected directly with the [im_kernel](#) module.

NOTE

The system logger may need to be disabled or reconfigured to collect logs with [im_kernel](#). To completely disable syslogd on FreeBSD, run `service syslogd onestop` and `sysrc syslogd_enable=NO`. On OpenBSD, run `rcctl stop syslogd` and `rcctl disable syslogd`.

Example 135. Collecting Kernel Logs

This configuration reads events from the kernel.

nxlog.conf

```
1 <Input kernel>
2   Module  im_kernel
3 </Input>
```

Local Syslog

Messages written to `/dev/log` can be collected with the [im_uds](#) module. Events written to file in Syslog format can be collected with [im_file](#). In both cases, the [xm_syslog](#) module can be used to parse the events. See the [Syslog](#) section for more information.

Example 136. Reading Syslog Messages From File

This example reads Syslog messages from `/var/log/messages` and parses them with the [parse_syslog\(\)](#) procedure.

nxlog.conf

```
1 <Extension _syslog>
2   Module  xm_syslog
3 </Extension>
4
5 <Input in>
6   Module  im_file
7   File    "/var/log/messages"
8   Exec    parse_syslog();
9 </Input>
```

Basic Security Mode (BSM) Auditing

The [im_bsm](#) module collects logs generated by the BSM auditing system.

NOTE

OpenBSD does not support BSM Auditing.

Example 137. Collecting BSM Audit Logs

This example reads BSM audit logs from the `/dev/auditpipe` device file.

nxlog.conf

```
1 <Input bsm>
2   Module      im_bsm
3   DeviceFile  /dev/auditpipe
4 </Input>
```

Process Accounting

The `im_acct` module can be used to gather details about who runs what processes.

Example 138. Reading Process Accounting Logs

This configuration turns on process accounting (using `/var/account/acct` as the log file) and watches for messages.

nxlog.conf

```
1 <Input acct>
2   Module    im_acct
3   AcctOn   TRUE
4   File     "/var/account/acct"
5 </Input>
```

File Integrity Monitoring

The `im_fim` module can be used to monitor files for changes.

Example 139. Monitoring File Integrity

This example monitors files in the `/etc` and `/srv` directories, generating events when files are modified or deleted. Files ending in `.bak` are excluded from the watch list.

nxlog.conf

```
1 <Input fim>
2   Module      im_fim
3   File        "/etc/*"
4   File        "/srv/*"
5   Exclude    "*.bak"
6   Digest      sha1
7   ScanInterval 3600
8 </Input>
```

Log Files

The `im_file` module can be used to collect events from log files.

Example 140. Reading From Log Files

This configuration reads messages from the `/opt/test/input.log` file. No parsing is performed; each line is available in the `$raw_event` field.

nxlog.conf

```
1 <Input in>
2   Module im_file
3   File   "/opt/test/input.log"
4 </Input>
```

Custom Programs

The `im_exec` module allows log data to be collected from custom external programs.

Example 141. Using an External Command

This example uses the `tail` command to read from a file.

NOTE

The `im_file` module should be used to read log messages from files. This example only demonstrates the use of the `im_exec` module.

nxlog.conf

```
1 <Input exec>
2   Module im_exec
3   Command /usr/bin/tail
4   Arg   -f
5   Arg   /var/log/messages
6 </Input>
```

Chapter 28. GNU/Linux

NXLog can collect various types of system logs on GNU/Linux platforms. See the [Supported Platforms](#) section for the list of supported GNU/Linux versions.

Kernel

The [im_kernel](#) module reads logs directly from the kernel log buffer. These logs can be parsed with [xm_syslog](#). See the [Linux System Logs](#) section.

Local Syslog

Messages written to [/dev/log](#) can be collected with the [im_uds](#) module. Events written to file in Syslog format can be collected with [im_file](#). In each case, the [xm_syslog](#) module can be used to parse the events. See the [Linux System Logs](#) and [Collecting and Parsing Syslog](#) sections for more information.

Linux Audit System

The [im_linuxaudit](#) module can be used to collect Audit System logs directly from the kernel without using auditd or temporary log files. Audit logs can also be collected from file with [im_file](#); or via the network with the Audit Dispatcher, the [audisp-remote](#) plugin, and [im_tcp](#). See [Linux Audit System](#) for more details.

Process Accounting

The [im_acct](#) module can be used to gather details about who runs what processes. This overlaps with Audit System logging.

File Integrity Monitoring

The [im_fim](#) module can be used to monitor files for changes.

Log Files

The [im_file](#) module can be used to collect events from log files.

Log Databases

Events can be read from databases with the [im_dbi](#), [im_oci](#), and [im_odbc](#) modules.

Custom Programs and Scripts

The [im_exec](#) module allows log data to be collected from custom external programs. The [im_perl](#), [im_python](#) and [im_ruby](#) modules can also be used to implement integration with custom data sources or sources that are not supported out-of-the-box.

Chapter 29. Apple macOS

NXLog can collect various types of system logs on the macOS platform. See the [Supported Platforms](#) section for the list of supported Apple macOS versions.

Kernel

Logs from the kernel can be collected directly with the [im_kernel](#) module. This requires disabling syslogd. Alternatively, kernel logs can be collected via the local log file with [im_file](#); see [Local Syslog](#) below.

Example 142. Collecting Kernel Logs Directly

This configuration uses the [im_kernel](#) module to read events directly from the kernel (via [/dev/klog](#)). This requires that syslogd be disabled as follows:

1. Unload the daemon.

```
$ sudo launchctl unload /System/Library/LaunchDaemons/com.apple.syslogd.plist
```

2. Rename plist to keep syslogd from starting again at the next reboot.

```
$ sudo mv /System/Library/LaunchDaemons/com.apple.syslogd.plist \
/System/Library/LaunchDaemons/com.apple.syslogd.plist.disabled
```

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input kernel>
6   Module im_kernel
7   Exec parse_syslog_bsd();
8 </Input>
```

Local Syslog

Events written to file in Syslog format can be collected with [im_file](#). The [xm_syslog](#) module can be used to parse the events. See the [Syslog](#) section for more information.

Example 143. Reading Syslog Messages From File

This configuration file collects system logs from [/var/log/system.log](#). This method does not read from [/dev/klog](#) directly, so it is not necessary to disable syslogd.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input in>
6   Module im_file
7   File  "/var/log/system.log"
8   Exec  parse_syslog();
9 </Input>
```

Apple System Logs Files

The [im_file](#) and [xm_asl](#) modules can be used to collect and parse Apple System Log ([*.asl](#)) files.

Example 144. Reading and Parsing Apple System Logs

This example reads events from `input.asl` and parses them with the `xm_asl` parser.

`nxlog.conf`

```
1 <Extension asl_parser>
2   Module xm_asl
3 </Extension>
4
5 <Input in>
6   Module im_file
7   # Example: "/var/log/asl/*"
8   File "foo/input.asl"
9   InputType asl_parser
10  Exec delete($EventReceivedTime);
11 </Input>
```

Basic Security Mode (BSM) Auditing

The `im_bsm` module collects logs directly from the BSM auditing system.

Example 145. Collecting BSM Audit Logs From the Kernel

This configuration reads BSM audit logs directly from the kernel with the `im_bsm` module.

`nxlog.conf`

```
1 Group wheel
2
3 <Input bsm>
4   Module im_bsm
5   File /dev/auditpipe
6 </Input>
```

Alternatively, BSM logs can be read from the log files.

Example 146. Reading BSM Audit Logs From File

This configuration reads from the BSM audit log files with `im_file` and parses the events with `xm_bsm`.

`nxlog.conf`

```
1 Group wheel
2
3 <Extension bsm_parser>
4   Module xm_bsm
5 </Extension>
6
7 <Input bsm>
8   Module im_file
9   File '/var/audit/*'
10  InputType bsm_parser
11 </Input>
```

Process Accounting

The `im_acct` module can be used to gather details about who runs what processes.

Example 147. Reading Process Accounting Logs

With this configuration file, NXLog will enable process accounting to the specified file and reads events from it.

nxlog.conf

```
1 Group    wheel
2
3 <Input acct>
4     Module   im_acct
5     File     '/var/log/acct'
6     AcctOn  TRUE
7 </Input>
```

File Integrity Monitoring

The [im_fim](#) module can be used to monitor files for changes.

Example 148. Monitoring File Integrity

This configuration watches for changes to files and directories under [/bin](#) and [/usr/bin/](#).

nxlog.conf

```
1 <Input fim>
2     Module      im_fim
3     File        "/bin/*"
4     File        "/usr/bin/*"
5     ScanInterval 3600
6 </Input>
```

Log Files

The [im_file](#) module can be used to collect events from log files.

Example 149. Reading From Log Files

This configuration uses the [im_file](#) module to read events from the specified log file.

nxlog.conf

```
1 <Input in>
2     Module   im_file
3     File    "/foo/in.log"
4 </Input>
```

Custom Programs

The [im_exec](#) module allows log data to be collected from custom external programs.

Example 150. Using an External Command

This example uses the **tail** command to read from a file.

NOTE

The **im_file** module should be used to read log messages from files. This example only demonstrates the use of the **im_exec** module.

nxlog.conf

```
1 <Input systemlog>
2   Module im_exec
3   Command /usr/bin/tail
4   Arg    -f
5   Arg    /var/log/system.log
6 </Input>
```

Chapter 30. Oracle Solaris

NXLog can collect various types of system logs on the Solaris platform. See the [Supported Platforms](#) section for the list of supported Oracle Solaris versions.

Local Syslog

Events written to file in Syslog format can be collected with the [im_file](#) module and parsed with the [xm_syslog](#) module. See the [Syslog](#) section for more information.

Basic Security Mode (BSM) Auditing

The [im_bsm](#) module collects logs generated by the BSM auditing system.

Process Accounting

The [im_acct](#) module can be used to gather details about who runs what processes.

File Integrity Monitoring

The [im_fim](#) module can be used to monitor files for changes.

Log Files

The [im_file](#) module can be used to collect events from log files.

Custom Programs

The [im_exec](#) module allows log data to be collected from custom external programs.

Chapter 31. Microsoft Windows

NXLog can collect various types of system logs on the Windows platform. See the [Supported Platforms](#) section for the list of supported Microsoft Windows versions.

Windows EventLog

See the [Windows EventLog](#) section, which covers both local and remote event collection with the `im_msvisalog`, `im_wseventing`, `im_mseventlog`, and `im_wmi` modules.

Windows Event Tracing (ETW)

Events logged through ETW can be collected with the `im_etw` module. This includes events logged to the Analytical and Debug logs.

Sysmon

Many additional audit events can be generated with the Sysmon utility, including process creation, system driver loading, network connections, and modification of file creation timestamps. These events are written to the EventLog. See the [Sysmon](#) section for more information.

Windows Performance Counters

The `im_winperfcount` module can be used for collecting data such as CPU and memory usage.

File Integrity Monitoring

The `im_fim` module can be used to monitor files for changes.

Registry Monitoring

The Windows Registry can be monitored for changes; see the `im_regmon` module.

Log Files

The `im_file` module can be used to collect events from log files.

Log Databases

Events can be read from databases with the `im_odbc` module. Some products write logs into SQL Server databases; see the [Microsoft System Center Operations Manager](#) section for example.

Custom Programs

The `im_exec` module allows log data to be collected from custom external programs.

Microsoft IIS

IIS can be configured to write logs in W3C format, which can be read with `im_file` and parsed with `xm_w3c` or `xm_csv`. Other formats can be parsed with other methods; see [Microsoft IIS](#).

Microsoft SQL Server

Log messages can be collected from the Microsoft SQL Server error log files with the `im_file` module. See [Microsoft SQL Server](#).

Integration

Chapter 32. Apache HTTP Server

The Apache HTTP Server provides very comprehensive and flexible logging capabilities. A brief overview is provided in the following sections; see the [Log Files](#) section of the Apache HTTP Server Documentation for more detailed information about configuring logging.

32.1. Error Log

Apache error logging is controlled by the `ErrorLog`, `ErrorLogFormat`, and `LogLevel` directives. The error log can be parsed by NXLog with a regular expression.

Example 151. Using the Apache Error Log

The following directives enable error logging of all messages at or above the "informational" severity level, in the specified format, to the specified file. The **ErrorLogFormat** defined below is equivalent to the default (which includes the timestamp, the module producing the message, the event severity, the process ID, the thread ID, the client address, and the detailed error message).

apache2.conf

```
LogLevel info
ErrorLogFormat "[%{u}t] [%-m:%l] [pid %P:tid %T] [client %a] %M"
ErrorLog /var/log/apache2/error.log
```

Following is a typical log message generated by the Apache HTTP Server, an NXLog configuration for parsing it, and the resulting JSON.

Log Sample

```
[Tue Aug 01 07:17:44.496832 2017] [core:info] [pid 15019:tid 140080326108928] [client 192.168.56.1:60154] AH00128: File does not exist: /var/www/html/notofile.html
```

nxlog.conf

```
1 <Input apache_error>
2   Module im_file
3   File   '/var/log/apache2/error.log'
4 <Exec>
5     if $raw_event =~ /(?x)^[\S+ ([^\]]+)]\ \[(\S+):(\S+)\]\ \[pid\ (\d+):
6           tid\ (\d+)\]\ \([client\ (\S+)\]\ )?(.+)$/
7     {
8       $EventTime = parsedate($1);
9       $ApacheModule = $2;
10      $ApacheLogLevel = $3;
11      $ApachePID = $4;
12      $ApacheTID = $5;
13      if $7 != '' $ClientAddress = $7;
14      $Message = $8;
15    }
16  </Exec>
17 </Input>
```

Output Sample

```
{
  "EventReceivedTime": "2017-08-01T07:17:45.641190+02:00",
  "SourceModuleName": "apache_error",
  "SourceModuleType": "im_file",
  "EventTime": "2017-08-01T07:17:44.496832+02:00",
  "ApacheModule": "core",
  "ApacheLogLevel": "info",
  "ApachePID": "15019",
  "ApacheTID": "140080317716224",
  "ClientAddress": "192.168.56.1:60026",
  "Message": "AH00128: File does not exist: /var/www/html/notofile.html"
}
```

32.2. Access Log

The access log file and format are configured with the **LogFormat** and **CustomLog** directives. The **LogFormat** directive is used to define a format, while the **CustomLog** directive configures logging to a specified file in one of the defined formats. Multiple **CustomLog** directives can be used to enable logging to multiple files.

There are several options for handling logging when using virtual hosts. The examples below, when specified in the main server context (not in a `<VirtualHost>` section), will log all requests exactly as with a single-host server. The `%v` format string can be added, if desired, to log the name of the virtual server responding to the request. Alternatively, the `CustomLog` directive can be specified inside a `<VirtualHost>` section, and in this case only the requests served by that virtual server will be logged to the file.

NOTE Pre-defined format strings for the Common Log and Combined Log Formats may be included by default. These pre-defined formats may use `%0` (the total sent including headers) instead of the standard `%b` (the size of the requested file) in order to allow detection of partial requests.

Example 152. Using the Common Log Format for the Access Log

The `LogFormat` directive below creates a format named `common` that corresponds to the Common Log Format. The second directive configures the Apache HTTP Server to write entries to the `access_log` file in the `common` format.

apache2.conf

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common  
CustomLog /var/log/apache2/access_log common
```

Example 153. Using the Combined Log Format for the Access Log

The following directives will configure the Apache HTTP Server to write entries to the `access_log` file in the Combined Log Format.

apache2.conf

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined  
CustomLog /var/log/apache2/access_log combined
```

NXLog configuration examples for parsing these access log formats can be found in the [Common & Combined Log Formats](#) section.

Chapter 33. Apache Tomcat

Apache Tomcat provides flexible [logging](#) that can be configured for different transports and formats.

Example 154. Collecting Apache Tomcat Logs

Here is a log sample consisting of three events. The log message of the second event spans multiple lines.

Apache Tomcat Log Example

```
2001-01-25 17:31:42,136 INFO [org.nxlog.somepackage.Class] - single line↵
2001-01-25 17:41:16,268 ERROR [org.nxlog.somepackage.Class] - Error retrieving names: ; nested
exception is:↵
    java.net.ConnectException: Connection refused↵
AxisFault↵
    faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException↵
    faultSubcode:↵
    faultString: java.net.ConnectException: Connection refused↵
    faultActor:↵
    faultNode:↵
    faultDetail:↵
        {http://xml.apache.org/axis/}stackTrace:java.net.ConnectException: Connection refused↵
2001-01-25 17:57:38,469 INFO [org.nxlog.somepackage.Class] - third log message↵
```

This can be very useful to filter messages by the emitting Java class, for example.

nxlog.conf

```
1 <Input in>
2     Module im_file
3     File   "/var/log/tomcat6/catalina.out"
4     <Exec>
5         if $raw_event =~ /(?x)^(\d{4}\-\d{2}\-\d{2}\ \ \d{2}:\d{2}:\d{2}),\d{3}
6             \ (\S+)\ \[(\S+)\]\ \-\ (.*)/
7         {
8             $log4j.time = parsedate($1);
9             $log4j.level = $2;
10            $log4j.class = $3;
11            $log4j.msg = $4;
12        }
13    </Exec>
14 </Input>
```

To parse and process multi-line messages such as the above, see [Parsing Multi-Line Messages](#).

Chapter 34. APC Automatic Transfer Switch

The APC Automatic Transfer Switch (ATS) is capable of sending its logs to a remote Syslog destination via UDP.

Log Sample

Date	Time	Event
03/26/2017	16:20:55	Automatic Transfer Switch: Communication established.
03/26/2017	16:20:45	System: Warmstart.
03/26/2017	16:19:13	System: Detected an unauthorized user attempting to access the SNMP interface from 192.168.15.11.

The ATS is an independent device, so if there more than one installed in a particular environment the configuration below must be applied to each device individually. For more details about configuring APC ATS logging, go to the [APC Support Site](#) and select the product name or part number.

NOTE

The steps below have been tested on AP7700 series devices and should work for other ATS models also.

1. Configure NXLog for receiving log entries via UDP (see the [example](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from the device.
3. Configure Syslog logging on the ATS using either the web interface or the command line. See the following sections.

Example 155. Receiving Logs from APC ATS

The following examples shows the ATS logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10    Module im_udp
11    Host   0.0.0.0
12    Port   514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/apc.log"
19    Exec   to_json();
20 </Output>
```

Logs like the example at the beginning of the chapter will produce output as follows.

Output Sample

```
{  
    "MessageSourceAddress": "192.168.15.22",  
    "EventReceivedTime": "2017-03-26 17:03:27",  
    "SourceModuleName": "in_syslog_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 23,  
    "SyslogFacility": "LOCAL7",  
    "SyslogSeverityValue": 7,  
    "SyslogSeverity": "DEBUG",  
    "SeverityValue": 1,  
    "Severity": "DEBUG",  
    "Hostname": "192.168.15.22",  
    "EventTime": "2017-03-26 16:04:18",  
    "SourceName": "System",  
    "Message": "Detected an unauthorized user attempting to access the SNMP interface from  
192.168.15.11. 0x0004"  
}  
  
{  
    "MessageSourceAddress": "192.168.15.22",  
    "EventReceivedTime": "2017-03-26 17:20:04",  
    "SourceModuleName": "in_syslog_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 23,  
    "SyslogFacility": "LOCAL7",  
    "SyslogSeverityValue": 7,  
    "SyslogSeverity": "DEBUG",  
    "SeverityValue": 1,  
    "Severity": "DEBUG",  
    "Hostname": "192.168.15.22",  
    "EventTime": "2017-03-26 16:20:54",  
    "SourceName": "System",  
    "Message": "Warmstart. 0x0002"  
}  
  
{  
    "MessageSourceAddress": "192.168.15.22",  
    "EventReceivedTime": "2017-03-26 17:20:04",  
    "SourceModuleName": "in_syslog_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 23,  
    "SyslogFacility": "LOCAL7",  
    "SyslogSeverityValue": 7,  
    "SyslogSeverity": "DEBUG",  
    "SeverityValue": 1,  
    "Severity": "DEBUG",  
    "Hostname": "192.168.15.22",  
    "EventTime": "2017-03-26 16:20:55",  
    "Message": "Automatic Transfer Switch: Communication established. 0x0C05"  
}
```

34.1. Configuring via the Web Interface

1. Log in to the web panel.
2. Go to **Network > Syslog**.
3. Enable **Syslog**.
4. Select the **Facility**.

5. Add up to four Syslog servers and a port for each.
6. Map the **Local Severity** to the **Syslog Severity** as required.

Syslog Configuration		
General Setting		
Syslog:	Enabled	
Facility:	Local7	
Syslog Server	Server IP/Domain Name	Port
Server 1:	192.168.15.251	514
Server 2:	0.0.0.0	514
Server 3:	0.0.0.0	514
Server 4:	0.0.0.0	514
Local Severity	Map to Syslog's Severity	
Severe:	Debug	
Warning:	Debug	
Informational:	Debug	
None:	Debug	

Buttons: Apply, Cancel

7. Click [**Apply**].

34.2. Configuring via the Command Line

1. Log in to the ATS via Telnet.
2. Type **2** and then **9** to go to the Syslog settings.
3. Type **1** to configure the Syslog settings.
4. Type **1** to enable Syslog.
5. Type **2** to configure the Syslog facility.
6. Type **3** to save the changes.
7. Press **ESC** to go one level up.
8. Select one of the four Syslog server slots.
9. Type **1** to set the Syslog server IP address.
10. Type **2** to change set the UDP port number.
11. Type **3** to apply the changes.
12. Press **ESC** to go one level up.
13. Type **6** to map the local severity to the Syslog severity.
14. Use options from **1** to **4** to choose the mapping.
15. Type **5** to accept the changes.

Example 156. ATS Syslog Settings

The following shows the Syslog settings screen, which is shown after completing step 2 above.

```
----- Syslog -----  
  
Syslog Settings          Severity Mapping  
-----  
Syslog : Enabled          Severe : DEBUG          Info: DEBUG  
Facility: LOCAL7          Warning: DEBUG         None: DEBUG  
  
#  Syslog Server Port    IP  
-----  
1  514                  192.168.15.251  
2  514                  0.0.0.0  
3  514                  0.0.0.0  
4  514                  0.0.0.0  
  
1- Settings  
2- Server 1  
3- Server 2  
4- Server 3  
5- Server 4  
6- Severity Mapping  
  
<ESC>- Back, <ENTER>- Refresh, <CTRL-L>- Event Log  
> 1
```

Chapter 35. ArcSight Common Event Format (CEF)

NXLog can be configured to collect or forward logs in Common Event Format (CEF). NXLog Enterprise Edition provides the [xm_cef](#) module for parsing and generating CEF.

CEF is a text-based log format developed by ArcSight™ and used by HP ArcSight™ products. It uses Syslog as transport. The full format includes a Syslog header or "prefix", a CEF "header", and a CEF "extension". The extension contains a list of key-value pairs. Standard key names are provided, and user-defined extensions can be used for additional key names. In some cases, CEF is used with the Syslog header omitted.

CEF Syntax

```
Jan 11 10:25:39 host CEF:Version|Device Vendor|Device Product|Device Version|Device Event Class  
ID|Name|Severity|[Extension]←
```

Log Sample

```
Oct 12 04:16:11 localhost CEF:0|nxlog.org|nxlog|2.7.1243|Executable Code was Detected|Advanced  
exploit detected|100|src=192.168.255.110 spt=46117 dst=172.25.212.204 dpt=80←
```

For more information, see HPE Security's [Implementing ArcSight Common Event Format](#) guide.

35.1. Collecting and Parsing CEF

NXLog Enterprise Edition can be configured to collect and parse CEF logs with the [xm_cef](#) module.

The ArcSight™ Logger can be configured to send CEF logs via TCP with the following steps.

1. Log in to the Logger control panel.
2. Browse to **Configuration > Data > Forwarders**.
3. Click **Add** to create a new Forwarder:
 - Name: **nxlog**
 - Type: **TCP Forwarder**
 - Type of Filter: **Unified Query**
4. Click **Next** to proceed to editing the new Forwarder:
 - Query: (define as required)
 - IP/Host: (enter the IP address or hostname of the system running NXLog)
 - Port: **1514**
5. Click **Save**.

Example 157. Receiving CEF Logs

With this configuration, NXLog will collect CEF logs via TCP, convert to plain JSON format, and save to file.

nxlog.conf

```
1 <Extension _cef>
2     Module xm_cef
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Extension _syslog>
10    Module xm_syslog
11 </Extension>
12
13 <Input logger_tcp>
14     Module im_tcp
15     Host   0.0.0.0
16     Port   1514
17     Exec   parse_syslog(); parse_cef($Message);
18 </Input>
19
20 <Output json_file>
21     Module om_file
22     File   '/var/log/json'
23     Exec   to_json();
24 </Output>
25
26 <Route r>
27     Path   logger_tcp => json_file
28 </Route>
```

35.2. Generating and Forwarding CEF

NXLog Enterprise Edition can be configured to generate and forward CEF logs with the [xm_cef](#) module.

The ArcSight™ Logger can be configured to receive CEF logs via TCP with the following steps.

1. Log in to the Logger control panel.
2. Browse to **Configuration > Data > Receivers** in the navigation menu.
3. Click **Add** to create a new Receiver:
 - Name: **nxlog**
 - Type: **CEF TCP Receiver**
4. Click **Next** to proceed to editing the new Receiver:
 - Port: **574**
 - Encoding: **UTF-8**
 - Source Type: **CEF**
5. Click **Save**.

Example 158. Sending CEF Logs

With this configuration, NXLog will read Syslog logs from file, convert them to CEF, and forward them to the ArcSight Logger via TCP. Default values will be used for the CEF header unless corresponding fields are defined in the event record (see the [to_cef\(\)](#) procedure in the Reference Manual for a list of fields).

nxlog.conf

```
1 <Extension _cef>
2     Module xm_cef
3 </Extension>
4
5 <Extension _syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input messages_file>
10    Module im_file
11    File   '/var/log/messages'
12    Exec   parse_syslog();
13 </Input>
14
15 <Output logger_tcp>
16    Module om_tcp
17    Host   192.168.1.1
18    Port   574
19    Exec   $Message = to_cef(); to_syslog_bsd();
20 </Output>
21
22 <Route r>
23     Path   messages_file => logger_tcp
24 </Route>
```

35.3. Using xm_csv and xm_kvp

Because NXLog Community Edition does not include the [xm_cef](#) module, the [xm_csv](#) and [xm_kvp](#) modules may be used instead to handle CEF logs.

WARNING The [xm_csv](#) and [xm_kvp](#) modules may not always correctly parse or generate CEF logs.

Example 159. Using CEF with NXLog Community Edition

Here, the [xm_csv](#) module is used to parse the pipe-delimited CEF header, while the [xm_kvp](#) module is used to parse the space-delimited key-value pairs in the CEF extension. The required extension configurations are shown below.

nxlog.conf Extensions

```

1 <Extension cef_header>
2   Module      xm_csv
3   Fields      $Version, $Device_Vendor, $Device_Product, $Device_Version, \
4               $Signature_ID, $Name, $Severity, $_Extension
5   Delimiter   |
6   QuoteMethod None
7 </Extension>
8
9 <Extension cef_extension>
10  Module      xm_kvp
11  KVDelimiter '='
12  KVPDelimiter ' '
13  QuoteMethod None
14 </Extension>
15
16 <Extension syslog>
17  Module      xm_syslog
18 </Extension>
```

For CEF input, use an input instance like this one.

nxlog.conf Input

```

1 <Input in>
2   Module  im_tcp
3   Host    0.0.0.0
4   Port    1514
5   <Exec>
6     parse_syslog();
7     cef_header->parse_csv($Message);
8     cef_extension->parse_kvp($_Extension);
9   </Exec>
10 </Input>
```

For CEF output, use an output instance like this one.

nxlog.conf Output

```

1 <Output out>
2   Module  om_tcp
3   Host    192.168.1.1
4   Port    574
5   <Exec>
6     $_Extension = cef_extension->to_kvp();
7     $Version = 'CEF:0';
8     $Device_Vendor = 'NXLog';
9     $Device_Product = 'NXLog';
10    $Device_Version = '';
11    $Signature_ID = '0';
12    $Name = '-';
13    $Severity = '';
14    $Message = cef_header->to_csv();
15    to_syslog_bsd();
16  </Exec>
17 </Output>
```

Chapter 36. AWS CloudWatch

AWS CloudWatch is a set of cloud monitoring services. The Amazon CloudWatch Logs service can be used to collect log data from Elastic Compute Cloud (EC2), CloudTrail, Route 53, and other sources. NXLog is able to retrieve CloudWatch Logs log streams, by either pulling the data via API or listening for a push from Lambda.

See the [CloudWatch documentation](#) for more information about configuring and using AWS CloudWatch Logs.

36.1. Pulling via API

NXLog can be configured to poll for logs via the API. This setup is suitable when a short delay in log collection is acceptable. To configure NXLog to perform a periodical pull of logs from the cloud, please follow the steps below.

1. First of all, a service account to access the logs must be created. In the AWS web interface, go to **Services > IAM**.
2. Click the **Users** option in the left-side panel.
3. Click the **[Add user]** button.
4. Provide a **User name**, for example **nxlog**. Tick the checkbox to allow **Programmatic access** to this account.

The screenshot shows the 'Add user' wizard in the AWS IAM console. It consists of three tabs at the top: 'Details' (highlighted in blue), 'Permissions', and 'Review'. The 'Details' tab has a step indicator '1' above it. The main area is titled 'Set user details' and contains a 'User name*' field with 'nxlog' entered. Below it is a link '+ Add another user'. The 'Permissions' tab has a step indicator '2' above it. The 'Review' tab has a step indicator '3' above it.

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name* nxlog

+ Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type* **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

AWS Management Console access
Enables a **password** that allows users to sign-in to the AWS Management Console.

5. Choose to **Attach existing policies directly** and select the **CloudWatchLogsReadOnly** policy. Click **[Next: Review]**.

Attach one or more existing policies directly to the users or create a new policy. [Learn more](#)

Create policy Refresh

Filter: Policy type ▾ Logs

	Policy name ▾	Type	Attachments ▾	Description
<input type="checkbox"/>	▶ AmazonAPIGatewayPush...	AWS managed	0	Allows API Gateway to push logs to CloudWatch Logs
<input type="checkbox"/>	▶ AmazonDMSCloudWatchL...	AWS managed	0	Provides access to upload DMS replication logs to CloudWatch Logs
<input type="checkbox"/>	▶ AWSOpsWorksCloudWat...	AWS managed	0	Enables OpsWorks instances with CloudWatch Logs
<input type="checkbox"/>	▶ CloudWatchLogsFullAccess	AWS managed	1	Provides full access to CloudWatch Logs
<input checked="" type="checkbox"/>	▶ CloudWatchLogsReadOnl...	AWS managed	0	Provides read only access to CloudWatch Logs

6. Click the [**Create user**] button.
7. Save access keys for this user and [**Close**].

Example 160. Using a Python Script for CloudWatch API Input

In this example, the `im_python` module is used to pull the logs from Amazon cloud. A similar script could be written in Perl or Ruby.

```
nxlog.conf
1 define PYDIR /opt/nxlog/lib/nxlog/modules/input/python
2
3 <Input py>
4   Module      im_python
5   PythonCode  %PYDIR%/im_python.py
6 </Input>
```

NOTE

In order to use the script below, the AWS SDK for Python (Boto 3) must be installed first. This can be accomplished by running `pip install boto3`.

aws-logs.py

```
import nxlog, boto3, json, time

class LogReader:

    def __init__(self, time_interval):
        client = boto3.client('logs', region_name='eu-central-1')

        self.lines = ""
        all_streams = []
        group_name = '<ENTER GROUP NAME HERE>'
```

```

#query CloudWatch for all log streams in the group
stream_batch = client.describe_log_streams(logGroupName=group_name)
all_streams += stream_batch[ 'logStreams' ]
start_time = int(time.time()-time_interval)*1000
end_time = int(time.time())*1000

while 'nextToken' in stream_batch:
    stream_batch = client.describe_log_streams(
        logGroupName=group_name, nextToken=stream_batch[ 'nextToken' ])
    all_streams += stream_batch[ 'logStreams' ]
    nxlog.log_debug(str(len(all_streams)))

#get log data from all available streams
for stream in all_streams:
    #get first log batch (up to 10,000 log events)
    logs_batch = client.get_log_events(logGroupName=group_name,
                                         logStreamName=stream[ 'logStreamName' ],
                                         startTime=start_time,
                                         endTime=end_time)
    #write events from the first batch in JSON format
    self.json_dump(logs_batch, group_name, stream[ 'logStreamName' ])

    #get next log batches till all the data is collected
    while 'nextToken' in logs_batch:
        logs_batch = client.get_log_events(
            logGroupName=group_name, logStreamName=stream[ 'logStreamName' ],
            startTime=start_time, endTime=end_time,
            nextToken=logs_batch[ 'nextToken' ])
        self.json_dump(logs_batch, group_name, stream[ 'logStreamName' ])
    nxlog.log_debug('Pulling logs: ' + gettime(start_time) + ' - ' +
                    gettime(end_time) + '\n')

def json_dump(self, cloudwatch_logs, group_name, stream_name):
    for event in cloudwatch_logs[ 'events' ]:
        event.update({ 'group': group_name, 'stream': stream_name })
        self.lines += json.dumps(event) + '\n'

def getlogs(self):
    if not self.lines:
        return None
    return self.lines

def gettime(time_milliseconds):
    return time.strftime('%Y-%m-%d %H:%M:%S',
                        time.localtime(time_milliseconds/1000))

def read_data(module):
    # log pull time interval in seconds
    time_interval = 300

    module[ 'reader' ] = LogReader(time_interval)
    reader = module[ 'reader' ]
    logdata = module.logdata_new()
    line = reader.getlogs()

    if line:
        logdata.set_field( 'raw_event', line)
        logdata.post()
        nxlog.log_debug("Data posted")

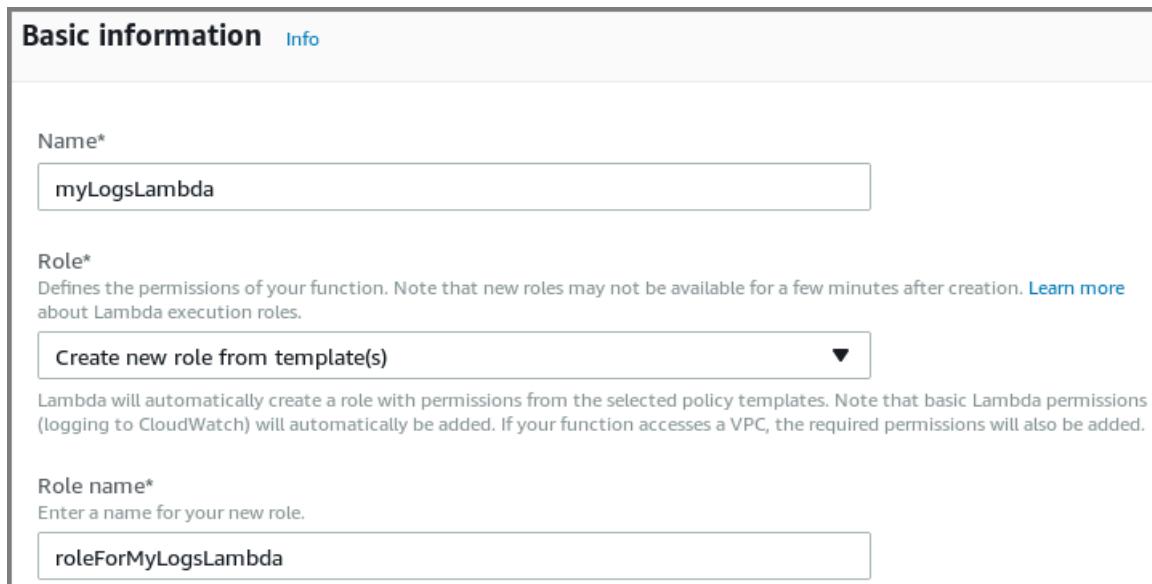
```

```
module.set_read_timer(time_interval)  
  
nxlog.log_info("INIT SCRIPT")
```

36.2. Pushing Log Data With Lambda

Using a push model follows an event-driven computing approach and allows for low latency. In this scenario, the AWS Lambda function sends log data in JSON format with the HTTP POST method. NXLog is used as a collector for these requests.

1. In the AWS web interface, go to **Services > Lambda**.
2. Click the **[Create function]** button.
3. Click the **[Author from scratch]** button.
4. Provide the name for your function and choose to create a new role from template. Enter a name for the role associated with Lambda function. Then click the **[Create function]** button.



The screenshot shows the 'Basic information' section of the AWS Lambda function creation wizard. It includes fields for Name (set to 'myLogsLambda'), Role (set to 'Create new role from template(s)'), and Role name (set to 'roleForMyLogsLambda'). A note below the role selection explains that Lambda will automatically create a role with permissions from selected policy templates, including basic Lambda permissions and CloudWatch logging.

Basic information	
Name*	myLogsLambda
Role*	Create new role from template(s)
Role name*	roleForMyLogsLambda

5. Select the **Python** runtime and choose to **Upload a .ZIP file**. Upload an archive with the code (and optionally with certificates, if needed). Change the **Handler** name to the `file_name.function_name` format (for example `lambda_function.lambda_handler`). Click **[Save]**.

▼ Function code

i Your Lambda function "myLogsLambda" cannot be edited inline since the file name specified in the handler does not in your deployment package.

⚠ This function contains external libraries. Uploading a new file will override these libraries.

Code entry type

Upload a .ZIP file

Runtime

Python 3.6

Handler [Info](#)

lambda_function.lambda_handler

Function package*

U Upload

For files larger than 10 MB, consider uploading via S3.

6. From the **Configuration** tab, change to the **Triggers** tab. Click [**+ Add trigger**].
7. Choose **CloudWatch Logs** as a trigger for the Lambda function. Select the log group that should be forwarded and provide a **Filter Name**. Then click [**Submit**].

Add trigger

Configure your Lambda function myLogsLambda to respond to events from the selected trigger. Click on the box below select your trigger type.



Log Group

Please select the CloudWatch Logs log group that serves as the event source. Log Events sent to the log group will trigger your lambda function with the contents of the logs received.

/aws/lambda/myLogs

Filter Name

Choose a name for your filter.

myLambdaTrigger

Filter Pattern

Enter an optional Filter Pattern.

Lambda will add the necessary permissions for Amazon CloudWatch Logs to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Enable trigger

Enable the trigger now, or create it in a disabled state for testing (recommended).

Example 161. Lambda Collection via HTTPS Input

In this example, the [im_http](#) module is used to receive the logs from Amazon cloud. The Lambda function and associated PEM certificates should be uploaded to the cloud in a zip archive, via the AWS management console.

nxlog.conf

```
1 <Input http>
2   Module      im_http
3   ListenAddr  127.0.0.1
4   Port        8080
5   HTTPSCertFile    %CERTDIR%/server-cert.pem
6   HTTPSCertKeyFile  %CERTDIR%/server-key.pem
7   HTTPSChainFile   %CERTDIR%/ca.pem
8   HTTPSRequireCert TRUE
9   HTTPSAllowUntrusted FALSE
10 </Input>
```

lambda_function.py

```
import json, base64, zlib, ssl, http.client

print('Loading function')

def lambda_handler(event, context):
    compressed_logdata = base64.b64decode(event['awslogs']['data'])
    logdata = zlib.decompress(compressed_logdata, 16+zlib.MAX_WBITS)
    context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
    context.load_verify_locations("ca.pem")
    context.load_cert_chain("client.pem")

    conn = http.client.HTTPSConnection("<HOST>:<PORT>", context=context)
    conn.set_debuglevel(3)

    headers = {"Content-type": "application/json"}

    conn.request('POST', "", logdata, headers)
    conn.close()
```

NOTE

Please refer to Python's [ssl module documentation](#) for more details regarding the `SSLContext.load_cert_chain(certfile, keyfile=None, password=None)` function.

Chapter 37. Azure Operations Management Suite

The Azure Operations Management Suite (OMS) is a set of Microsoft cloud services providing log management, backup, automation, and high availability features. Azure Log Analytics is the part of OMS used for log collection, correlation, and analysis.

See the [Azure OMS](#) and [Log Analytics](#) documentation for more information about configuring and using Azure OMS and its log management service.

37.1. Forwarding Data to Log Analytics

In order to configure NXLog to forward log data to the Log Analytics service, complete the following steps.

1. Log in to the Azure portal and go to the **Log Analytics** service (for instance by typing the service name into the search bar).
2. Select an existing **OMS Workspace** or create a new one by clicking the **[+ Add]** button.
3. From the **Management** section in the main workspace screen, click **OMS Portal**.

The screenshot shows the Azure Log Analytics workspace settings page. On the left, there's a sidebar with a search bar at the top, followed by sections for 'OMS Workspace' (selected), 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. Below that is a 'SETTINGS' section with 'Locks', 'Automation script', and 'Advanced settings'. The main content area has a header with 'OMS Portal', 'Delete', and 'Upgrade Summary Log' buttons. A 'Essentials' tab is selected, showing details like Resource group (rg-nxlog), Status (Active), Location (East US), Subscription name (nxlog), and Management services (Operations logs). Below this is a 'Management' section with three buttons: 'Overview', 'Log Search', and 'OMS Portal'.

4. In the **Microsoft Operations Management Suite** click the settings icon in the top right corner, navigate to **Settings > Connected Sources > Linux Servers**, and copy the **WORKSPACE ID** and **PRIMARY KEY**. Save them to a text file; they are needed for API access.

Windows Servers

Linux Servers

Azure Storage

System Center

Windows Telemetry

Linux Servers
Attach any Linux server or client.

0 LINUX COMPUTERS CONNECTED

Download Agent for Linux

You'll need the Workspace ID and Key to install the agent.

WORKSPACE ID

PRIMARY KEY

Regenerate

SECONDARY KEY

Regenerate

5. Enable **Custom Logs**. As of this writing it is a preview feature, available under **Settings > Preview Features > Custom Logs**.

FEATURE NAME	DESCRIPTION	
Custom Logs	Configure OMS to collect custom text log files from your local environment and create searchable events from them.	Enabled Disabled

Example 162. Forwarding to Azure Log Analytics With a Python Script

The `om_exec` module can be used in conjunction with a Python script to forward log data to the Log Analytics service.

NOTE NXLog must have permission to execute the script.

`nxlog.conf`

```

1 define PYCODEDIR /opt/nxlog/lib/nxlog/modules/output/python
2
3 <Extension _json>
4     Module xm_json
5 </Extension>
6
7 <Output oms>
8     Module om_exec
9     Command %PYCODEDIR%/oms-pipe.py
10    Exec    to_json();
11 </Output>
```

The following Python code implements the OMS API and performs the REST API call.

oms-pipe.py

```
import requests
import datetime
import hashlib
import hmac
import base64
import fileinput

# Update the customer ID to your Operations Management Suite workspace ID
customer_id = '<cid>'

# For the shared key, use either the primary or the secondary Connected Sources client
# authentication key
shared_key = "<skey>"

# The log type is the name of the event that is being submitted
log_type = 'STDIN_PY'

# Build the API signature
def build_signature(customer_id, shared_key, date, content_length, method, content_type,
resource):
    x_headers = 'x-ms-date:' + date
    string_to_hash = method + "\n" + str(content_length) + "\n" + content_type + "\n" +
x_headers + "\n" + resource
    bytes_to_hash = bytes(string_to_hash).encode('utf-8')
    decoded_key = base64.b64decode(shared_key)
    encoded_hash = base64.b64encode(hmac.new(decoded_key, bytes_to_hash, digestmod=hashlib.
sha256).digest())
    authorization = "SharedKey {}:{}".format(customer_id, encoded_hash)
    return authorization

# Build and send a request to the POST API
def post_data(customer_id, shared_key, body, log_type):
    method = 'POST'
    content_type = 'application/json'
    resource = '/api/logs'
    rfc1123date = datetime.datetime.utcnow().strftime('%a, %d %b %Y %H:%M:%S GMT')
    content_length = len(body)
    signature = build_signature(customer_id, shared_key, rfc1123date, content_length, method,
content_type, resource)
    uri = 'https://' + customer_id + '.ods.opinsights.azure.com' + resource + '?api-
version=2016-04-01'

    headers = {
        'Content-Type': content_type,
        'Authorization': signature,
        'Log-Type': log_type,
        'x-ms-date': rfc1123date
    }

    response = requests.post(uri, data=body, headers=headers)
    if (response.status_code >= 200 and response.status_code <= 299):
        print 'Accepted'
    else:
        print "Response code: {}".format(response.status_code)

for body in fileinput.input():
    post_data(customer_id, shared_key, body, log_type)
```

37.2. Downloading Data From Log Analytics

NXLog can be used to download data from Log Analytics service. This can be achieved by using the OMS API, for example with a Python script. Enable the API by following these steps.

1. Register an application in **Azure Active Directory**.
2. Generate an access key for this application.
3. Under your **Subscription**, go to **Access control (IAM)** and assign the **Log Analytics Reader** role to this application.

Detailed instructions on this topic can be found in the [Azure documentation](#).

Example 163. Collecting Logs From Azure Log Analytics With a Python Script

The `im_python` module can be used in conjunction with a Python script to periodically collect log data from the Log Analytics service.

`nxlog.conf`

```
1 define PYDIR /opt/nxlog/lib/nxlog/modules/input/python
2
3 <Input oms>
4     Module      im_python
5     PythonCode  %PYDIR%/oms-download.py
6 </Input>
```

The following Python code implements the OMS API and performs the REST API call. In order to authenticate, the Subscription ID, Tenant ID, Application ID, and Application Key are required.

NOTE

The Tenant ID can be found as "Directory ID" under the Azure Active Directory **Properties** tab.

`oms-download.py`

```
import datetime
import json
import requests

import adal
import nxlog

class LogReader:

    def __init__(self, time_interval):
        # Details of workspace. Fill in details for your workspace.
        resource_group = '<YOUR_RESOURCE_GROUP>'
        workspace = '<YOUR_WORKSPACE>'

        # Details of query. Modify these to your requirements.
        query = "Type=*" 
        end_time = datetime.datetime.utcnow()
        start_time = end_time - datetime.timedelta(seconds=time_interval)
        num_results = 100000 # If not provided, a default of 10 results will be used.

        # IDs for authentication. Fill in values for your service principal.
        subscription_id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
        tenant_id = 'xxxxxxxx-xxxx-xxxx-xxx-xxxxxxxxxxxx'
        application_id = 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'
        application_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

```
# URLs for authentication
authentication_endpoint = 'https://login.microsoftonline.com/'
resource   = 'https://management.core.windows.net/'

# Get access token
context = adal.AuthenticationContext('https://login.microsoftonline.com/' + tenant_id)
token_response = context.acquire_token_with_client_credentials
('https://management.core.windows.net/', application_id, application_key)
access_token = token_response.get('accessToken')

# Add token to header
headers = {
    "Authorization": 'Bearer ' + access_token,
    "Content-Type": 'application/json'
}

# URLs for retrieving data
uri_base = 'https://management.azure.com'
uri_api = 'api-version=2015-11-01-preview'
uri_subscription = 'https://management.azure.com/subscriptions/' + subscription_id
uri_resourcegroup = uri_subscription + '/resourcegroups/' + resource_group
uri_workspace = uri_resourcegroup +
'/providers/Microsoft.OperationalInsights/workspaces/' + workspace
uri_search = uri_workspace + '/search'

#store log data for NXLog here
self.lines = ""

# Build search parameters from query details
search_params = {
    "query": query,
    "top": num_results,
    "start": start_time.strftime('%Y-%m-%dT%H:%M:%S'),
    "end": end_time.strftime('%Y-%m-%dT%H:%M:%S')
}

# Build URL and send post request
uri = uri_search + '?' + uri_api
response = requests.post(uri, json=search_params, headers=headers)

# Response of 200 if successful
if response.status_code == 200:

    # Parse the response to get the ID and status
    data = response.json()
    search_id = data["id"].split("/")
    id = search_id[len(search_id)-1]
    status = data["__metadata"]["Status"]

    # If status is pending, then keep checking until complete
    while status == "Pending":

        # Build URL to get search from ID and send request
        uri_search = uri_search + '/' + id
        uri = uri_search + '?' + uri_api
        response = requests.get(uri, headers=headers)

        # Parse the response to get the status
        data = response.json()
```

```
status = data["__metadata"]["Status"]
else:

    # Request failed
    print(response.status_code)
    response.raise_for_status()

    print("Total records:" + str(data["__metadata"]["total"]))
    print("Returned top:" + str(data["__metadata"]["top"]))

#write a JSON dump of all events
for event in data['value']:
    self.lines += json.dumps(event) + '\n'

def getlogs(self):
    if not self.lines:
        return None
    return self.lines

def read_data(module):
    # log pull time interval in seconds
    time_interval = 300

    module['reader'] = LogReader(time_interval)
    reader = module['reader']
    logdata = module.logdata_new()
    line = reader.getlogs()

    if line:
        logdata.set_field('raw_event', line)
        logdata.post()
        nxlog.log_debug("Data posted")

    module.set_read_timer(time_interval)

nxlog.log_info("INIT SCRIPT")
```

Chapter 38. Bro Network Security Monitor

NXLog can be configured to collect events generated by the [Bro Network Security Monitor](#), a powerful open source Intrusion Detection System (IDS) and network traffic analysis framework. The Bro engine captures traffic and converts it to a series of high-level events. These events are then analyzed according to customizable policies. Bro supports real-time alerts, data logging for further investigation, and automatic program execution for detected anomalies. Bro is able to analyze different protocols, including HTTP, FTP, SMTP, and DNS; as well as run host and port scans, detect signatures, and discover syn-floods.

38.1. About Bro Logs

Bro creates different log files in order to record network activities such as files transferred over the network, SSL sessions, and HTTP requests. By default, BRO provides 60 different log files.

Table 55. A Few of Bro's Default Log Files

File	Description
conn.log	TCP/UDP/ICMP connections
dhcp.log	DHCP leases
dns.log	DNS activity
files.log	Summaries of files transferred over the network
ftp.log	FTP activity
http.log	HTTP requests and replies
smtp.log	SMTP transactions
ssl.log	SSL/TLS handshake information
weird.log	Unexpected network-level activity

Bro produces human-readable logs in a format similar to W3C. Each log file uses a different set of fields.

dns.log Sample

```
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path dns
#open 2018-01-03-11-44-20
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto trans_id rtt query
qclass qclass_name qtype qtype_name rcode rcode_name AA TC RD RA Z answers TTLs
rejected
#types time string addr port addr port enum count interval string count
string count string count string bool bool bool bool count vector[string]
vector[interval] bool
1514951060.504593 CWKugb3ukgbFv1yhRf 192.168.56.101 59572 192.168.56.1 137 udp 23266 -
WORKGROUP 1 C_INTERNET 33 NBSTAT 0 NOERROR F F F 0 - - F
1514951060.504593 CQINoQ18vBSIBaLi09 192.168.56.101 49062 192.168.56.1 137 udp 7805 -
- - - - 0 NOERROR F F F F 0 - - F
1514951060.504593 CthpDaR4iKLHt13Qi 192.168.56.101 49062 192.168.56.255 137 udp 7805 -
WORKGROUP 1 C_INTERNET 32 NB - - F F T F 1 - - F
```

For more information about Bro logging, see the [Bro Manual](#).

38.2. Parsing Bro Logs

NXLog Enterprise Edition can parse Bro logs with the [xm_w3c](#) module.

NOTE The following configurations have been tested with Bro version 2.5.2.

Example 164. Using xm_w3c to Parse Bro Logs

This configuration reads Bro logs from a directory, parses with [xm_w3c](#), and writes out events in JSON format.

nxlog.conf

```
1 <Extension _json>
2     Module      xm_json
3 </Extension>
4
5 <Extension w3c_parser>
6     Module      xm_w3c
7 </Extension>
8
9 <Input bro_in>
10    Module     im_file
11    File       '/usr/local/bro/logs/current/*.log'
12    InputType  w3c_parser
13 </Input>
14
15 <Output bro_file>
16    Module     om_file
17    File       '/tmp/bro_logs'
18    Exec       to_json();
19 </Output>
```

The [xm_w3c](#) module is recommended because it supports reading the field list from the W3C-style log file header. For NXLog Community Edition, the [xm_csv](#) module could be used instead to parse Bro logs. A separate instance of *xm_csv* must be configured for each log type.

Example 165. Using xm_csv to Parse Bro Logs

This example has separate `xm_csv` module instances for the DNS and DHCP log types. Additional CSV parsers could be added for the remaining Bro log types.

nxlog.conf

```
1 <Extension csv_parser_dns>
2     Module      xm_csv
3     Fields      ts, uid id.orig_h, id.orig_p, id.resp_h, id.resp_p, proto, \
4                  trans_id, rtt query, qclass, qclass_name, qtype, qtype_name, \
5                  rcode, rcode_name, AA, TC, RD, RA, Z, answers, TTLs, rejected
6     Delimiter   \t
7 </Extension>
8
9 <Extension csv_parser_dhcp>
10    Module      xm_csv
11    Fields      ts, uid, id.orig_h, id.orig_p, id.resp_h, id.resp_p, mac, \
12                  assigned_ip, lease_time, trans_id
13    Delimiter   \t
14 </Extension>
15
16 # xm_fileop provides the `file_basename()` function
17 <Extension _fileop>
18     Module      xm_fileop
19 </Extension>
20
21 <Input bro_in>
22     Module      im_file
23     File        '/usr/local/bro/logs/current/*.log'
24     <Exec>
25         if file_basename(file_name()) == 'dhcp.log'
26         {
27             csv_parser_dhcp->parse_csv();
28         }
29         else if file_basename(file_name()) == 'dns.log'
30         {
31             csv_parser_dns->parse_csv();
32         }
33         else
34         {
35             log_warning('Bro log type not supported, check configuration');
36         }
37     </Exec>
38 </Input>
```

Chapter 39. Brocade Switches

Brocade switches can be configured to send Syslog messages to a remote destination, UDP port 514.

Log Sample

```
2017/03/22-23:05:12, [SEC-1203], 113962, FID 128, INFO, fcsw1, Login information: Login successful  
via TELNET/SSH/RSH. IP Addr: admin2↔
```

The best way to configure a Brocade switch is with the command line interface. In the case of multiple switches running in redundancy mode, each device must be configured separately.

More details on configuring Brocade switches can be found in the Brocade [Document Library](#): search for a particular switch model and select **Installation & Configuration Guides** from the **Filter** list.

NOTE

The steps below have been tested with Brocade 4100 series switches and OS v6. Newer software versions may have additional capabilities, such as sending logs over TLS.

1. Configure NXLog for receiving Syslog entries via UDP (see the [example](#) below), then restart NXLog.
2. Make sure the NXLog agent is accessible from the switch.
3. Log in to the switch via SSH.
4. Run the following commands. Replace **LEVEL** with an integer corresponding to the desired Syslog *local* facility (see the example). Replace **IP_ADDRESS** with the address of the NXLog agent.

```
# syslogdfacility -l LEVEL  
# syslogdIpAdd IP_ADDRESS
```

Example 166. Sending Logs With local5 Facility

The following commands query the current Syslog facility and then set up Syslog logging to 192.168.6.143 with Syslog facility **local5**.

```
fcsw1:admin> syslogdfacility  
Syslog facility: LOG_LOCAL7  
fcsw1:admin> syslogdfacility -l 5  
Syslog facility changed to LOG_LOCAL5  
fcsw1:admin> syslogdIpAdd 192.168.6.143  
Syslog IP address 192.168.6.143 added
```

Example 167. Receiving Brocade Logs

This example shows Brocade switch logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10    Module im_udp
11    Host   0.0.0.0
12    Port   514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/brocade.log"
19    Exec   to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.15",
  "EventReceivedTime": "2017-03-22 20:23:58",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 21,
  "SyslogFacility": "LOCAL5",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2017-03-22 20:23:58",
  "Hostname": "192.168.5.15",
  "SourceName": "raslogd",
  "Message": "2017/03/22-23:05:12, [SEC-1203], 113962, WWN 10:00:00:05:1e:02:8e:fc | FID 128,
INFO, fcsw1, Login information: Login successful via TELNET/SSH/RSH. IP Addr: admin2"
}
```

Chapter 40. Checkpoint

The [im_checkpoint](#) module, provided by NXLog Enterprise Edition, can collect logs from Checkpoint devices over the OPSEC LEA protocol.

Example 168. Collecting CheckPoint LEA Logs

With the following configuration, NXLog will collect logs from CheckPoint devices over the LEA protocol and write them to file in JSON format.

nxlog.conf

```
1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Input checkpoint>
6   Module      im_checkpoint
7   Command     /opt/nxlog/bin/nx-im-checkpoint
8   LEAConfigFile /opt/nxlog/etc/lea.conf
9 </Input>
10
11 <Output file>
12   Module      om_file
13   File        'tmp/output'
14   Exec        $raw_event = to_json();
15 </Output>
16
17 <Route checkpoint_to_file>
18   Path        checkpoint => file
19 </Route>
```

Chapter 41. Cisco ACS

An example Syslog record from a Cisco Secure Access Control System (ACS) device looks like the following. For more information, refer to the [Syslog Logging Configuration Scenario](#) chapter in the Cisco Configuration Guide.

Log Sample

```
<38>Oct 16 21:01:29 10.0.1.1 CisACS_02_FailedAuth 1k1fg93nk 1 0 Message-Type=Authen failed,User-  
Name=John,NAS-IP-Address=10.0.1.2,AAA Server=acs01←
```

Example 169. Collecting From Cisco Secure ACS

The following configuration file instructs NXLog to accept Syslog messages on UDP port 1514. The payload is parsed as Syslog and then the ACS specific fields are extracted. The output is written to file in JSON format.

nxlog.conf

```
1 <Extension _json>  
2     Module xm_json  
3 </Extension>  
4  
5 <Extension _syslog>  
6     Module xm_syslog  
7 </Extension>  
8  
9 <Input in>  
10    Module im_udp  
11    Host   0.0.0.0  
12    Port   1514  
13    <Exec>  
14        parse_syslog_bsd();  
15        if ( $Message =~ /^CisACS_(\d\d)_($+ )($+ )($+ )($+ )(.*)$/ )  
16        {  
17            $ACSCategoryNumber = $1;  
18            $ACSCategoryName = $2;  
19            $ACSMessagId = $3;  
20            $ACSTotalSegments = $4;  
21            $ACSSegmentNumber = $5;  
22            $ACSMessag = $6;  
23            if ( $ACSMessag =~ /Message-Type=(^\,+)/ ) $ACSMessagType = $1;  
24            if ( $ACSMessag =~ /User-Name=(^\,+)/ ) $AccountName = $1;  
25            if ( $ACSMessag =~ /NAS-IP-Address=(^\,+)/ ) $ACSNASIPAddress = $1;  
26            if ( $ACSMessag =~ /AAA Server=(^\,+)/ ) $ACSAAServer = $1;  
27        }  
28        else log_warning("Does not match: " + $raw_event);  
29    </Exec>  
30 </Input>  
31  
32 <Output out>  
33     Module om_file  
34     File   "tmp/output.txt"  
35     Exec   to_json();  
36 </Output>
```

Chapter 42. Cisco ASA

Cisco Adaptive Security Appliance (ASA) devices are capable of sending their logs to a remote Syslog destination via TCP or UDP. When sending logs over the network, it is recommended to use TCP as the more reliable protocol. With UDP there is a potential to lose entries, especially when there is a high volume of messages.

Log Sample

```
Apr 15 2017 00:21:14 192.168.12.1 : %ASA-5-111010: User 'john', running 'CLI' from IP 0.0.0.0,
executed 'dir disk0:/dap.xml'↵
Apr 15 2017 00:22:27 192.168.12.1 : %ASA-4-313005: No matching connection for ICMP error message:
icmp src outside:81.24.28.226 dst inside:72.142.17.10 (type 3, code 0) on outside interface.
Original IP payload: udp src 72.142.17.10/40998 dst 194.153.237.66/53.↵
Apr 15 2017 00:22:42 192.168.12.1 : %ASA-3-710003: TCP access denied by ACL from
179.236.133.160/8949 to outside:72.142.18.38/23.↵
```

For more details about configuring Syslog on Cisco ASA, check the [Cisco configuration guide](#) for the ASA or Adaptive Security Device Manager (ASDM) version in use.

NOTE

The steps below have been tested with ASA 9.x and ASDM 7.x, but should work for other versions also.

42.1. Forwarding Cisco ASA Logs Over TCP

1. Configure NXLog for receiving Syslog via TCP (see the examples below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from each of the ASA devices being configured.
3. Set up Syslog logging using either the command line or ASDM. See the following sections.

Example 170. Receiving Cisco ASA Logs

This example shows Cisco ASA logs as received and processed by NXLog.

```
nxlog.conf
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module xm_json
7 </Extension>
8
9 <Input in_syslog_tcp>
10  Module im_tcp
11  Host 0.0.0.0
12  Port 1514
13  Exec parse_syslog();
14 </Input>
15
16 <Output file>
17  Module om_file
18  File "/var/log/asa.log"
19  Exec to_json();
20 </Output>
```

Logs like the sample at the beginning of the chapter will result in the following output.

Output Sample

```
{  
    "MessageSourceAddress": "192.168.12.1",  
    "EventReceivedTime": "2017-04-15 00:19:53",  
    "SourceModuleName": "in_syslog_tcp",  
    "SourceModuleType": "im_tcp",  
    "SyslogFacilityValue": 20,  
    "SyslogFacility": "LOCAL4",  
    "SyslogSeverityValue": 5,  
    "SyslogSeverity": "NOTICE",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "Hostname": "192.168.12.1",  
    "EventTime": "2017-04-15 00:21:14",  
    "Message": "%ASA-5-111010: User 'john', running 'CLI' from IP 0.0.0.0, executed 'dir  
disk0:/dap.xml'"  
}  
  
{  
    "MessageSourceAddress": "192.168.12.1",  
    "EventReceivedTime": "2017-04-15 00:21:06",  
    "SourceModuleName": "in_syslog_tcp",  
    "SourceModuleType": "im_tcp",  
    "SyslogFacilityValue": 20,  
    "SyslogFacility": "LOCAL4",  
    "SyslogSeverityValue": 4,  
    "SyslogSeverity": "WARNING",  
    "SeverityValue": 3,  
    "Severity": "WARNING",  
    "Hostname": "192.168.12.1",  
    "EventTime": "2017-04-15 00:22:27",  
    "Message": "%ASA-4-313005: No matching connection for ICMP error message: icmp src  
outside:81.24.28.226 dst inside:72.142.17.10 (type 3, code 0) on outside interface. Original IP  
payload: udp src 72.142.17.10/40998 dst 194.153.237.66/53."  
}  
  
{  
    "MessageSourceAddress": "192.168.12.1",  
    "EventReceivedTime": "2017-04-15 00:21:21",  
    "SourceModuleName": "in_syslog_tcp",  
    "SourceModuleType": "im_tcp",  
    "SyslogFacilityValue": 20,  
    "SyslogFacility": "LOCAL4",  
    "SyslogSeverityValue": 3,  
    "SyslogSeverity": "ERR",  
    "SeverityValue": 4,  
    "Severity": "ERROR",  
    "Hostname": "192.168.12.1",  
    "EventTime": "2017-04-15 00:22:42",  
    "Message": "%ASA-3-71003: TCP access denied by ACL from 179.236.133.160/8949 to  
outside:72.142.18.38/23"  
}
```

The contents of the message can be parsed to extract additional fields.

Example 171. Extracting Additional Fields

The following configuration extracts the message ID from the \$Message field.

nxlog.conf

```
1 <Input in_syslog_tcp>
2   Module im_tcp
3   Host 0.0.0.0
4   Port 1514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /^(ASA)-(\d)-(\d{6}): (.*)$/
8     {
9       $ASASeverityNumber = $2;
10      $ASAMessageID = $3;
11      $ASAMessage = $4;
12    }
13  </Exec>
14 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.12.1",
  "EventReceivedTime": "2017-04-15 14:27:04",
  "SourceModuleName": "in_syslog_tcp",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 20,
  "SyslogFacility": "LOCAL4",
  "SyslogSeverityValue": 3,
  "SyslogSeverity": "ERR",
  "SeverityValue": 4,
  "Severity": "ERROR",
  "Hostname": " 192.168.12.1",
  "EventTime": "2017-04-15 14:28:26",
  "Message": "%ASA-3-710003: TCP access denied by ACL from 117.247.81.21/52569 to
outside:72.142.18.38/23",
  "ASASeverityNumber": "3",
  "ASAMessageID": "710003",
  "ASAMessage": "TCP access denied by ACL from 117.247.81.21/52569 to outside:72.142.18.38/23"
}
```

Further field extraction can be done based on message ID. Detailed information on existing IDs and their formats can be found in the [Cisco ASA Series Syslog Messages](#) book.

Example 172. Extracting Fields According to Message ID

The following NXLog configuration parses a very common firewall message: "TCP access denied by ACL".

nxlog.conf

```

1 <Input in_syslog_tcp>
2   Module im_tcp
3   Host 0.0.0.0
4   Port 1514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /(?x)^%ASA-3-710003:\ \ TCP\ access\ denied\ by\ ACL\ from
8       \ ([0-9.]*)\/([0-9]*)\ to\ outside:([0-9.]*)([0-9]*)/
9     {
10       $ASASeverityNumber = "3";
11       $ASAMessageID = "710003";
12       $ASAMessage = "TCP access denied by ACL";
13       $ASASrcIP = $1;
14       $ASASrcPort = $2;
15       $ASADstIP = $3;
16       $ASADstPort = $4;
17     }
18   </Exec>
19 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.12.1",
  "EventReceivedTime": "2017-04-15 15:10:20",
  "SourceModuleName": "in_syslog_tcp",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 20,
  "SyslogFacility": "LOCAL4",
  "SyslogSeverityValue": 3,
  "SyslogSeverity": "ERR",
  "SeverityValue": 4,
  "Severity": "ERROR",
  "Hostname": "192.168.12.1",
  "EventTime": "2017-04-15 15:11:43",
  "Message": "%ASA-3-710003: TCP access denied by ACL from 119.80.179.109/2083 to
outside:72.142.18.38/23",
  "ASASeverityNumber": "3",
  "ASAMessageID": "710003",
  "ASAMessage": "TCP access denied by ACL",
  "ASASrcIP": "119.80.179.109",
  "ASASrcPort": "2083",
  "ASADstIP": "72.142.18.38",
  "ASADstPort": "23"
}
```

42.1.1. Configuring via Command Line

To configure Cisco ASA Syslog logging from the command line, follow these steps.

1. Log in to the ASA device via SSH.
2. Enable logging.

```
# logging enable
```

3. In case of a High Availability (HA) pair, enable logging on the standby unit.

```
# logging standby
```

4. Specify the Syslog facility. Replace **FACILITY** with a number from 16 to 23, corresponding to **local0** through **local7** (the default is 20, or **local4**).

```
# logging facility FACILITY
```

5. Specify the severity level. Replace **LEVEL** with a number from 0 to 7. Use the maximum level for which messages should be generated (severity level 3 will produce messages for levels 3, 2, 1, and 0). The levels correspond to the [Syslog severities](#).

```
# logging trap LEVEL
```

6. Allow ASA to pass traffic when the Syslog server is not available.

```
# logging permit-hostdown
```

NOTE

If logs are being sent via TCP and this setting is not configured, ASA will stop passing traffic when the Syslog server is unavailable.

7. Configure the Syslog host. Replace **IP_ADDRESS** and **PORT** with the remote IP address and port that NXLog is listening on.

```
# logging host inside IP_ADDRESS tcp/PORT
```

NOTE

To enable SSL/TLS for connections to the NXLog agent, add **secure** at the end of the above command. The [im_ssl](#) module will need to be used when configuring NXLog.

Example 173. Redirecting Logs to 192.168.6.143

This command configures 192.168.6.143 as the Syslog host, with TCP port 1514.

```
# logging host inside 192.168.6.143 tcp/1514
```

8. Apply the configuration.

```
# write memory
```

42.1.2. Configuring via ASDM

To configure remote logging via ASDM, follow these steps.

1. Connect and log in to the GUI.
2. Go to **Configuration > Device Management > Logging > Logging Setup** and make sure **Enable Logging** is selected. In case of a High Availability (HA) pair, **Enable logging on the failover standby unit** should also be selected. Click **[Apply]**.

Configuration > Device Management > Logging > Logging Setup

<input checked="" type="checkbox"/> Enable logging	<input checked="" type="checkbox"/> Enable logging on the failover standby unit
<input type="checkbox"/> Send debug messages as syslogs	<input type="checkbox"/> Send syslogs in EMBLEM format

Logging to Internal Buffer

Specify the size of the internal buffer to which syslogs will be saved. When the buffer fills up, it will be overwritten.

Buffer Size: bytes

You can choose to save the buffer contents before the buffer is overwritten.

Save Buffer To: FTP Server

Flash

ASDM Logging

Specify the size of the queue for syslogs intended for viewing in ASDM.

Queue Size:

3. Go to **Syslog Setup** and specify the **Facility Code** (the default is 20). Click [**Apply**].

Configuration > Device Management > Logging > Syslog Setup

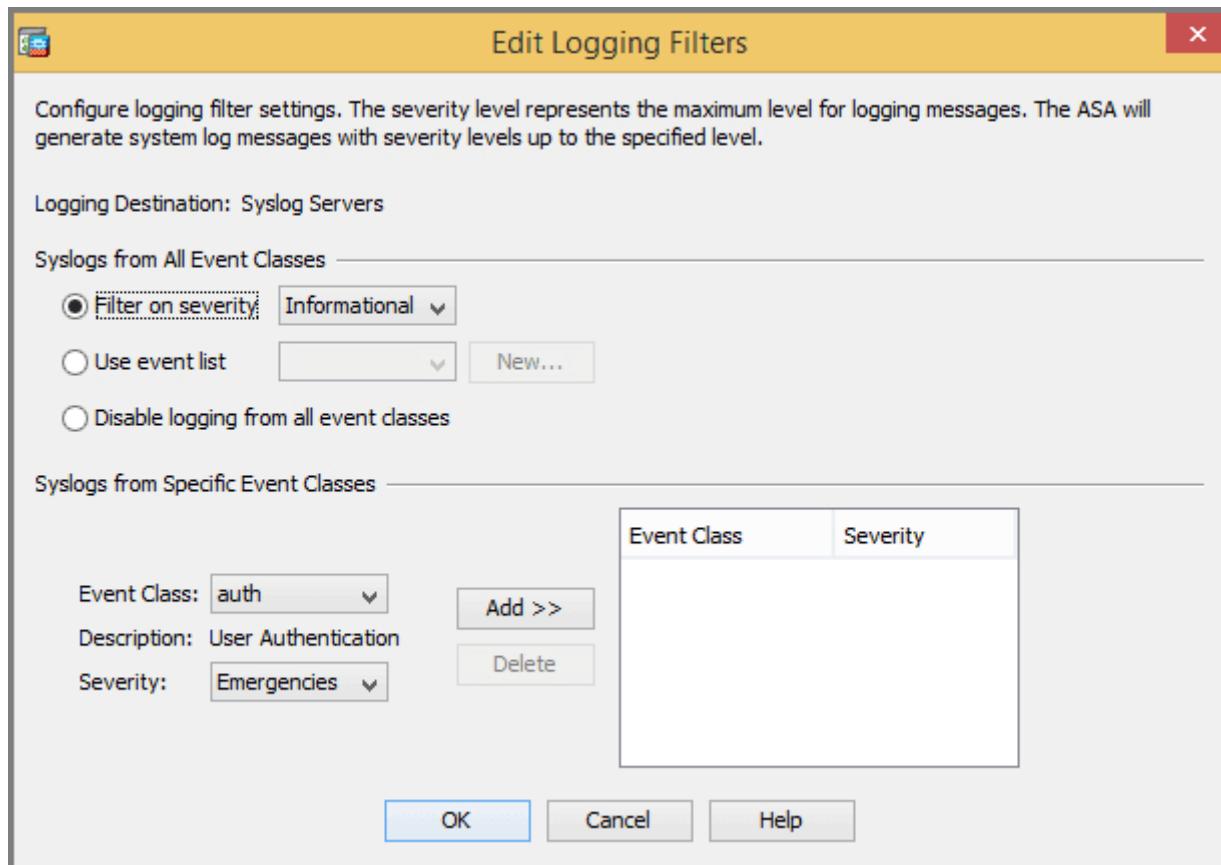
Syslog Format

Facility Code to Include in Syslogs:

Include timestamp in syslogs

Hide username if its validity cannot be determined

4. Go to **Logging Filters**, select **Syslog Servers**, click [**Edit**] and specify the severity level. Click [**OK**] and then [**Apply**].

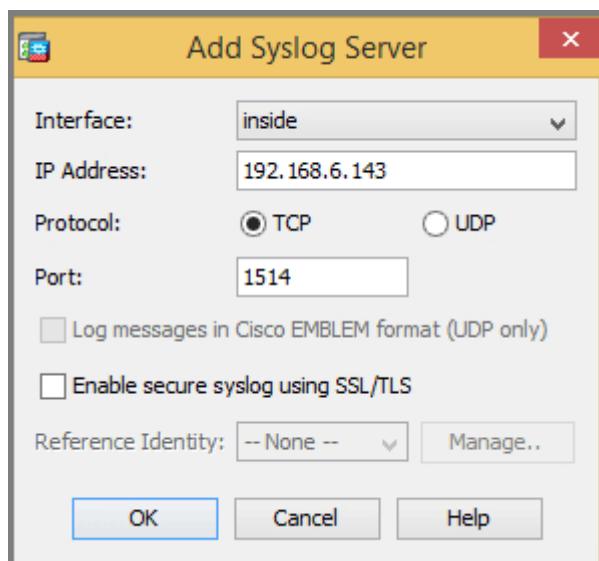


5. Go to **Syslog Servers** and select **Allow user traffic to pass when TCP syslog server is down**. Click [**Apply**].

NOTE

This setting is important to avoid downtime during TCP logging in case the Syslog server is unavailable.

6. Under **Syslog Servers**, click [**Add**] and specify the interface, remote IP address, protocol and port. Click [**OK**] and then [**Apply**].

**NOTE**

To enable SSL/TLS for connections to the NXLog agent, select the **Enable secure syslog using SSL/TLS** option. The [im_ssl](#) module will need to be used when configuring NXLog.

7. Click [**Save**] to save the configuration.

42.2. NetFlow From Cisco ASA

NetFlow is a protocol used by Cisco devices that provides the ability to send details about network traffic to a remote destination. NXLog is capable of receiving NetFlow logs. The steps below outline the configuration required to send information about traffic passing through Cisco ASA to NXLog via UDP.

1. Configure NXLog for receiving NetFlow via UDP/2162 (see the example below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from each of the ASA devices being configured.
3. Set up NetFlow logging on Cisco ASA, using either the command line or ASDM. See the following sections.

NOTE

The steps below have been tested with ASA 9.x and ASDM 7.x, but should work for other versions also.

Example 174. Receiving NetFlow Logs

This example shows NetFlow logs as received and processed by NXLog.

nxlog.conf

```

1 <Extension netflow>
2   Module      xm_netflow
3 </Extension>
4
5 <Extension _json>
6   Module      xm_json
7 </Extension>
8
9 <Input in_netflow_udp>
10  Module     im_udp
11  Host       0.0.0.0
12  Port       2162
13  InputType  netflow
14 </Input>
15
16 <Output file>
17  Module     om_file
18  File       "/var/log/netflow.log"
19  Exec       to_json();
20 </Output>
```

Output Sample

```
{
  "Version": 9,
  "SysUpTimeMilisec": 2374222958,
  "ExportTime": "2017-05-17 18:39:05",
  "TimeMsecStart": "2017-05-17 18:38:04",
  "Protocol": 6,
  "SourcePort": 64394,
  "DestPort": 443,
  "SourceIpV4Address": "192.168.13.37",
  "DestIpV4Address": "172.217.3.135",
  "inputSNMPIface": 4,
  "outputSNMPIface": 3,
  "ASAeventTime": "2017-05-17 18:39:05",
  "ASAconnID": 41834207,
  "FNF_ICMPCode": 0,
  "FNF_ICMPType": 0,
  "ASAevent": 1,
```

```

    "ASAextEvent": 0,
    "ASA_XlateSourcePort": 64394,
    "ASA_XlateDestPort": 443,
    "ASA_V4XlateSourceAddr": "72.142.18.38",
    "ASA_V4XlateDestAddr": "172.217.3.135",
    "ASA_IngressACL": "433a1af1a925365e00000000",
    "ASA_EgressACL": "000000000000000000000000",
    "ASA_UserName20": "",
    "MessageSourceAddress": "192.168.12.1",
    "EventReceivedTime": "2017-05-17 18:36:32",
    "SourceModuleName": "udpin",
    "SourceModuleType": "im_udp"
}
{
    "Version": 9,
    "SysUpTimeMilisec": 2374222958,
    "ExportTime": "2017-05-17 18:39:05",
    "TimeMsecStart": "2017-05-17 18:38:04",
    "Protocol": 17,
    "SourcePort": 65080,
    "DestPort": 443,
    "SourceIpV4Address": "192.168.13.37",
    "DestIpV4Address": "216.58.216.206",
    "inputSNMPIface": 4,
    "outputSNMPIface": 3,
    "ASAeventTime": "2017-05-17 18:39:05",
    "ASAconnID": 41834203,
    "FNF_ICMPCode": 0,
    "FNF_ICMPType": 0,
    "ASAevent": 1,
    "ASAextEvent": 0,
    "ASA_XlateSourcePort": 65080,
    "ASA_XlateDestPort": 443,
    "ASA_V4XlateSourceAddr": "72.142.18.38",
    "ASA_V4XlateDestAddr": "216.58.216.206",
    "ASA_IngressACL": "433a1af1a925365e00000000",
    "ASA_EgressACL": "000000000000000000000000",
    "ASA_UserName20": "",
    "MessageSourceAddress": "192.168.12.1",
    "EventReceivedTime": "2017-05-17 18:36:32",
    "SourceModuleName": "udpin",
    "SourceModuleType": "im_udp"
}
}

```

42.2.1. Configuring NetFlow via Command Line

To configure Cisco ASA NetFlow logging, follow these steps.

1. Log in to ASA via SSH.
2. Create a NetFlow destination. Replace **IP_ADDRESS** and **PORT** with the address and port that the NXLog agent is listening on.

```
# flow-export destination inside IP_ADDRESS PORT
```

3. Create an access list matching the traffic that needs to be logged. Replace **ACL_NAME** with a name for the access list. Replace **PROTOCOL**, **SOURCE_IP**, and **DESTINATION_IP** with appropriate values corresponding to the traffic to be matched.

```
# access-list ACL_NAME extended permit PROTOCOL SOURCE_IP DESTINATION_IP
```

4. Create a class map with the access list. Replace **ACL_NAME** with the access list name used in the previous step.

```
# class-map global-class  
# match access-list ACL_NAME
```

5. Add NetFlow destination to global policy. Replace **IP_ADDRESS** with the address that the NXLog agent is listening on.

```
# policy-map global_policy  
# class global-class  
# flow-export event-type all destination IP_ADDRESS
```

Example 175. Logging All Traffic to 192.168.6.143

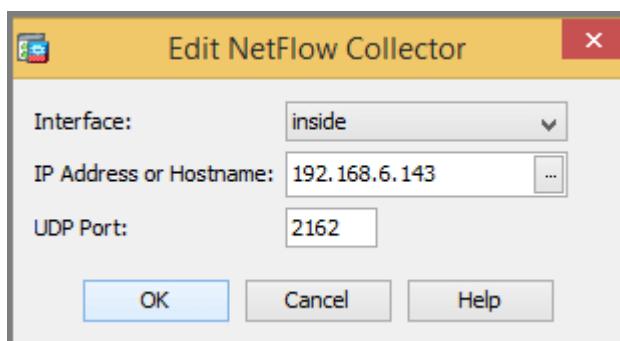
These commands enable NetFlow logging of all traffic to 192.168.6.143 via UDP port 2162.

```
# flow-export destination inside 192.168.6.143 2162  
# access-list global_mpc extended permit ip any any  
# class-map global-class  
# match access-list global_mpc  
# policy-map global_policy  
# class global-class  
# flow-export event-type all destination 192.168.6.143
```

42.2.2. Configuring NetFlow via ASDM

To configure Cisco ASA NetFlow logging via ASDM, follow these steps.

1. Connect and log in to the GUI.
2. Go to **Configuration > Device Management > Logging > NetFlow**.
3. Click [**Add**] and specify the interface, remote IP address, and port that the NXLog agent is listening on.



4. Go to **Configuration > Firewall > Service Policy Rules**.
5. Click [**Add**], switch to **Global**, and click [**Next**].

Add Service Policy Rule Wizard - Service Policy

Adding a new service policy rule requires three steps:

Step 1: Configure a service policy.

Step 2: Configure the traffic classification criteria for the service policy rule.

Step 3: Configure actions on the traffic classified by the service policy rule.

Create a Service Policy and Apply To: _____

Only one service policy can be configured per interface or at global level. If a service policy already exists, then you can add a new rule into the existing service policy. Otherwise, you can create a new service policy.

Interface: inside - (create new service policy)

Policy Name: inside-policy

Description: _____

Drop and log unsupported IPv6 to IPv6 traffic

Global - applies to all interfaces

Policy Name: global_policy *

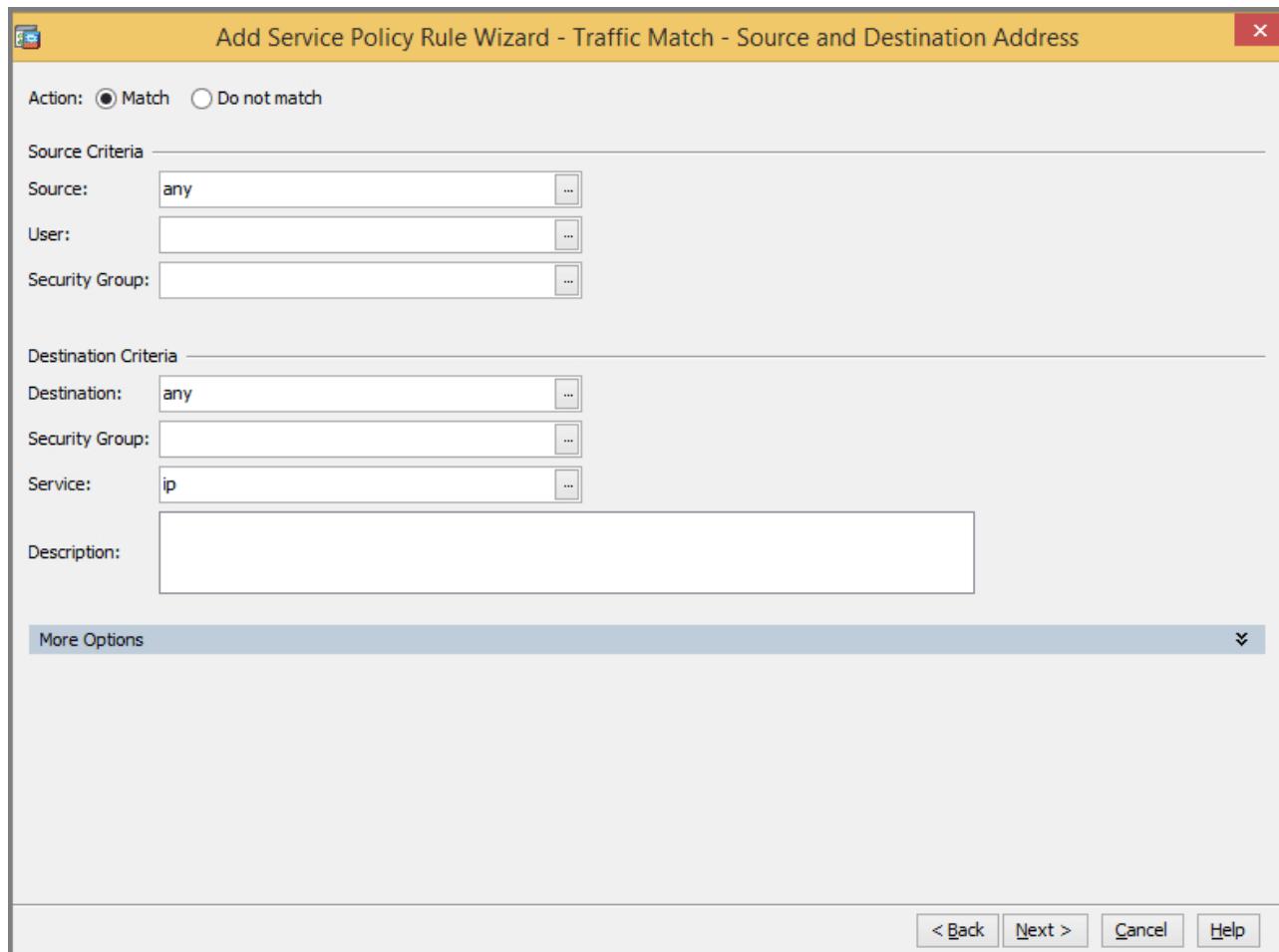
Description: _____

Drop and log unsupported IPv6 to IPv6 traffic

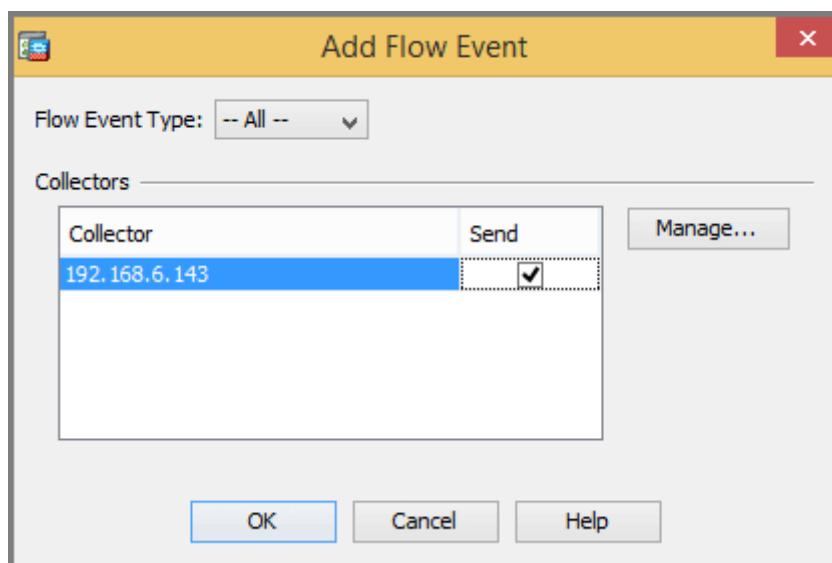
*Only one service policy is allowed. Existing service policy names cannot be changed.

< Back Cancel Help

6. Select **Source and Destination IP Address (uses ACL)** and click [**Next**].
7. Specify the source and destination criteria. The example below matches all traffic.



8. Go to the **NetFlow** tab and add the NetFlow destination created during the first step. Make sure the **Send** option is selected.



9. Click [OK] and [Finish].

Chapter 43. Common Event Expression (CEE)

NXLog can be configured to collect or forward logs in the Common Event Expression (CEE) format. [CEE](#) was developed by MITRE as an extension for Syslog, based on JSON. MITRE's work on CEE was discontinued in 2013.

Log Sample

```
Dec 20 12:42:20 syslog-relay serveapp[1335]: @cee:  
{"pri":10,"id":121,"appname":"serveapp","pid":1335,"host":"syslog-relay","time":"2011-12-  
20T12:38:05.123456-05:00","action":"login","domain":"app","object":"account","status":"success"}←
```

43.1. Collecting and Parsing CEE

NXLog can parse CEE with the [parse_json\(\)](#) procedure provided by the [xm_json](#) extension module.

Example 176. Collecting CEE Logs

With the following configuration, NXLog accepts CEE logs via TCP, parses the CEE-formatted **\$Message** field, and writes the logs to file in JSON format.

nxlog.conf

```

1 <Extension json>
2   Module xm_json
3 </Extension>
4
5 <Extension _syslog>
6   Module xm_syslog
7 </Extension>
8
9 <Input in>
10  Module im_tcp
11    Host 0.0.0.0
12    Port 1514
13    <Exec>
14      parse_syslog();
15      if $Message =~ /^@cee: ({.+})$/
16      {
17        $raw_event = $1;
18        parse_json();
19      }
20    </Exec>
21  </Input>
22
23 <Output out>
24   Module om_file
25   File '/var/log/json'
26   Exec to_json();
27 </Output>
```

Input Sample

```
Oct 13 14:23:11 myserver @cee: { "purpose": "test" }↔
```

Output Sample

```
{
  "EventReceivedTime": "2016-09-13 14:23:12",
  "SourceModuleName": "in",
  "SourceModuleType": "im_file",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "USER",
  "SyslogSeverityValue": 5,
  "SyslogSeverity": "NOTICE",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "myserver",
  "EventTime": "2016-09-13 14:23:11",
  "Message": "@cee: { \"purpose\": \"test\" }",
  "purpose": "test"
}
```

43.2. Generating and Forwarding CEE

NXLog can also generate CEE, using the **to_json()** procedure provided by the **xm_json** extension module.

Example 177. Generating CEE Logs

With this configuration, NXLog parses IETF Syslog input from file. The logs are then converted to CEE format and forwarded via TCP. The Syslog header data and IETF Syslog Structured-Data key/value list from the input are also included.

nxlog.conf

```
1 <Extension json>
2     Module xm_json
3 </Extension>
4
5 <Extension syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input in>
10    Module im_file
11    File   '/var/log/ietf'
12    Exec   parse_syslog();
13 </Input>
14
15 <Output out>
16    Module om_tcp
17    Host   192.168.1.1
18    Port   1514
19    Exec   $Message = '@cee: ' + to_json(); to_syslog_bsd();
20 </Output>
```

Input Sample

```
<13>1 2016-10-13T14:23:11.000000-06:00 myserver - - - [NXLOG@14506 Purpose="test"] This is a
test message.↵
```

Output Sample

```
<13>Oct 13 14:23:11 myserver @cee: {"EventReceivedTime":"2016-10-13
14:23:12", "SourceModuleName":"in", "SourceModuleType":"im_file", "SyslogFacilityValue":1, "SyslogF
acility":"USER", "SyslogSeverityValue":5, "SyslogSeverity":"NOTICE", "SeverityValue":2, "Severity":"
INFO", "EventTime":"2016-10-13 14:23:11", "Hostname":"myserver", "Purpose":"test", "Message":"This
is a test message."}↵
```

Chapter 44. Dell EqualLogic

Dell EqualLogic SAN systems are capable of sending logs to a remote Syslog destination via UDP.

In most environments, two or more EqualLogic units are configured as a single group. This allows storage capacity to be utilized from all devices, and the configuration of RAID levels across multiple drives and hardware platforms. In this case, Syslog configuration is performed from Group Manager and applies to all members.

Log Sample From a Group

```
AUDIT grpadmin 18-Mar-2017 20:13:01.508144 lab-array1 :CLI: Login to account grpadmin succeeded, using local authentication. User privilege is group-admin.←  
AUDIT grpadmin 18-Mar-2017 20:35:51.833836 lab-array1 :User action:volume select volume1 schedule create test type once start-time 06:30PM read-write max-keep 10 start-date 03/18/17 enable←  
11501:9173:lab-array1:MgmtExec:18-Mar-2017  
20:39:12.115208:snapshotDelete.cc:446:INFO:8.2.5:Successfully deleted snapshot volume1-2017-03-18-20:38:00.3.1.←
```

For more details about configuring logging on Dell EqualLogic PS series SANs, check the "Dell PS Series Configuration Guide" which can be downloaded from the [Dell EqualLogic Support Site](#) (a valid account is required).

NOTE The steps below have been tested with a Dell EqualLogic PS6000 series SAN.

1. Configure NXLog for receiving Syslog messages via UDP (see the examples below). Then restart NXLog.
 2. Make sure the NXLog agent is accessible from all devices in the group.
 3. Proceed with the logging configuration on EqualLogic, using either the Group Manager or the command line. See the following sections.

Example 178. Receiving Logs From EqualLogic

The following example shows EqualLogic logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10    Module im_udp
11    Host   0.0.0.0
12    Port   514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/equallogic.log"
19    Exec   to_json();
20 </Output>
```

Output Sample

```
{  
    "MessageSourceAddress": "192.168.10.43",  
    "EventReceivedTime": "2017-03-18 21:12:58",  
    "SourceModuleName": "in_syslog_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 16,  
    "SyslogFacility": "LOCAL0",  
    "SyslogSeverityValue": 6,  
    "SyslogSeverity": "INFO",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "EventTime": "2017-03-18 21:12:58",  
    "Hostname": "192.168.10.43",  
    "SourceName": "11517",  
    "Message": "380:netmgtd:18-Mar-2017  
21:13:19.415464:rcc_util.c:1032:AUDIT:grpadmin:25.7.0:CLI: Login to account grpadmin succeeded,  
using local authentication. User privilege is group-admin."  
}  
  
{  
    "MessageSourceAddress": "192.168.10.43",  
    "EventReceivedTime": "2017-03-18 20:35:31",  
    "SourceModuleName": "in_syslog_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 16,  
    "SyslogFacility": "LOCAL0",  
    "SyslogSeverityValue": 6,  
    "SyslogSeverity": "INFO",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "EventTime": "2017-03-18 20:35:31",  
    "Hostname": "192.168.10.43",  
    "SourceName": "11470",  
    "Message": "88:agent:18-Mar-2017 20:35:51.833836:echoCli.c:10611:AUDIT:grpadmin:22.7.0:User  
action:volume select volume1 schedule create test type once start-time 06:30PM read-write max-  
keep 10 start-date 03/18/17 enable"  
}  
  
{  
    "MessageSourceAddress": "192.168.10.43",  
    "EventReceivedTime": "2017-03-18 20:38:51",  
    "SourceModuleName": "in_syslog_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 16,  
    "SyslogFacility": "LOCAL0",  
    "SyslogSeverityValue": 6,  
    "SyslogSeverity": "INFO",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "EventTime": "2017-03-18 20:38:51",  
    "Hostname": "192.168.10.43",  
    "SourceName": "11502",  
    "Message": "103:agent:18-Mar-2017 20:39:12.124329:echoCli.c:10611:AUDIT:grpadmin:22.7.0:User  
action:volume select volume1 snapshot delete volume1-2017-03-18-20:38:00.3.1 "  
}
```

Example 179. Extracting Fields From the EqualLogic Logs

This configuration extracts additional fields from the messages.

nxlog.conf

```

1 <Input in_syslog_udp>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /(?x)^([0-9]*):([a-z]*):\d{2}-[a-zA-Z]{3}-\d{4}
8       \ \d{2}:\d{2}:\d{6}:([a-zA-Z.]*):[0-9]*:([a-zA-Z]*):
9         ([a-z]*):([0-9.]*):([a-zA-Z. ]*):(.*)$/
10    {
11      $EQLMsgSeq = $1;
12      $EQLMsgSrc = $2;
13      $EQLFile = $3;
14      $EQLMsgType = $4;
15      $EQLAccount = $5;
16      $EQLMsgID = $6;
17      $EQLEvent = $7;
18      $EQLMessage = $8;
19    }
20  </Exec>
21 </Input>
```

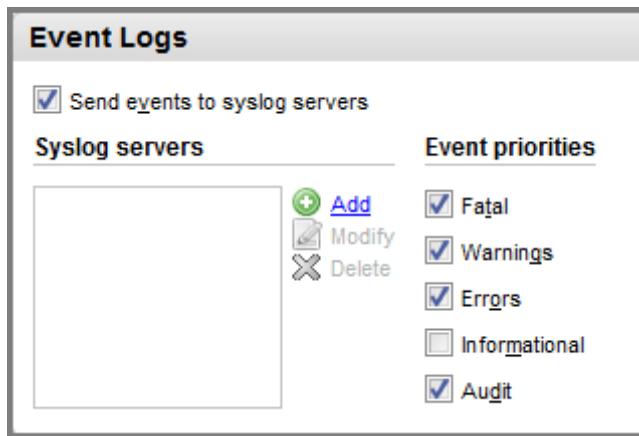
Output Sample

```
{
  "MessageSourceAddress": "192.168.10.43",
  "EventReceivedTime": "2017-04-15 16:55:48",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 16,
  "SyslogFacility": "LOCAL0",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2017-04-15 16:55:48",
  "Hostname": "192.168.10.43",
  "SourceName": "12048",
  "Message": "113:agent:15-Apr-2017 16:57:09.744470:echoCli.c:10611:AUDIT:grpadmin:22.7.0:User action:alerts select syslog priority fatal,error,warning,audit",
  "EQLMsgSeq": "113",
  "EQLMsgSrc": "agent",
  "EQLFile": "echoCli.c",
  "EQLMsgType": "AUDIT",
  "EQLAccount": "grpadmin",
  "EQLMsgID": "22.7.0",
  "EQLEvent": "User action",
  "EQLMessage": "alerts select syslog priority fatal,error,warning,audit"
}
```

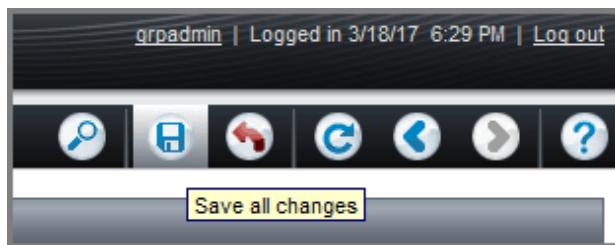
44.1. Configuring via the Group Manager

1. Log in to the Group Manager.

2. Go to **Group > Group Configuration > Notifications**.
3. Under the **Event Logs** section, make sure the **Send events to syslog servers** option is checked.
4. Select the required **Event priorities**.



5. Click [**Add**], enter the IP address of the NXLog agent, and click [**OK**].
6. Click the [**Save all changes**] button in the top left corner.



44.2. Configuring via the Command Line

1. Log in to the Group Manager via SSH.
2. Run the following commands. Replace **LEVELS** with a comma-separated list of event priorities. Available options include: **fatal**, **error**, **warning**, **info**, **audit**, and **none**. Replace **IP_ADDRESS** and **PORT** with the IP address and port that the NXLog agent is listening on.

```
# alerts select syslog priority LEVELS
# grpparams syslog-notify enable
# grpparams syslog-server-list IP_ADDRESS:PORT
```

Example 180. Sending Logs to the Specified Host

These commands will send all logs, except for Informational level, to 192.168.6.143 via the default UDP port 514.

```
# alerts select syslog priority fatal,error,warning,audit
# grpparams syslog-notify enable
# grpparams syslog-server-list 192.168.6.143
```

Chapter 45. Dell iDRAC

Integrated Dell Remote Access Controller (iDrac) is an interface that provides web-based or command line access to a server's hardware for management and monitoring purposes. This interface may be implemented as a separate expansion card (DRAC) or be integrated into the motherboard (iDRAC). In both cases it uses resources separate from the main server and is independent from the server's operating system.

Different server generations come with different versions of iDRAC. For example, PowerEdge R520, R620, or R720 servers have iDRAC7, while older models such as PowerEdge 1850 or 1950 come with iDRAC5. Remote Syslog via UDP is an option starting from iDRAC6.

NOTE An iDRAC Enterprise license is required to redirect logs to a remote Syslog destination.

Audit Log Sample

SeqNumber	= 1523↵
Message ID	= USR0030↵
Category	= Audit↵
AgentID	= RACLOG↵
Severity	= Information↵
Timestamp	= 2017-03-26 13:53:36↵
Message	= Successfully logged in using john, from 192.168.0.106 and GUI.↵
Message Arg 1	= john↵
Message Arg 2	= 192.168.0.106↵
Message Arg 3	= GUI↵
FQDD	= iDRAC.Embedded.1↵

For more details regarding iDRAC configuration, go to [Dell Support](#) and search for the server model or iDRAC version.

NOTE The steps below were tested with iDRAC7 but should work for newer versions as well.

1. Configure NXLog for receiving Syslog entries via UDP (see the examples below), then restart NXLog.
2. Make sure the NXLog agent is accessible from the management interface.
3. Configure iDRAC remote Syslog logging, using the web interface or the command line. See the following sections.

Example 181. Receiving iDRAC Logs

This example shows iDRAC logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10    Module im_udp
11    Host   0.0.0.0
12    Port   514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/idrac.log"
19    Exec   to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.50",
  "EventReceivedTime": "2017-03-26 13:52:48",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 21,
  "SyslogFacility": "LOCAL5",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2017-03-26 13:52:48",
  "Hostname": "192.168.5.50",
  "SourceName": "Severity",
  "Message": "Informational, Category: Audit, MessageID: USR0030, Message: Successfully logged in using john, from 192.168.0.106 and GUI."
}
```

Example 182. Extracting Additional Fields From iDRAC Logs

The following configuration extracts additional fields from the message.

nxlog.conf

```

1 <Input in_syslog_udp>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /(?x)^([a-zA-Z]*),\ Category:\ ([a-zA-Z]*),
8       \ MessageID:\ ([a-zA-Z0-9]*),\ Message:\ (.*)$/
9     {
10       $DracMsgLevel = $1;
11       $DracMscCategory = $2;
12       $DracMscID = $3;
13       $DracMessage = $4;
14     }
15   </Exec>
16 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.50",
  "EventReceivedTime": "2017-04-15 17:32:47",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 21,
  "SyslogFacility": "LOCAL5",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2017-04-15 17:32:47",
  "Hostname": "192.168.5.50",
  "SourceName": "Severity",
  "Message": "Informational, Category: Audit, MessageID: USR0030, Message: Successfully logged in using john, from 192.168.0.106 and GUI.",
  "DracMsgLevel": "Informational",
  "DracMscCategory": "Audit",
  "DracMscID": "USR0030",
  "DracMessage": "Successfully logged in using john, from 192.168.0.106 and GUI."
}
```

45.1. Configuring via the Web Interface

1. Log in to iDRAC.
2. Go to **Overview > Server > Alerts**.
3. Select the **Remote System Log** option for all required alert types.

Alerts and Remote System Log Configuration												▲ Back to Top
Category	Alert	Severity	Email	SNMP Trap	IPMI Alert	Remote System Log	WS Eventing	OS Log	Redfish Event	Action	Action	
			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
Audit	CMC	⚠	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	CMC	✗	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	Debug	ℹ	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	Debug	⚠	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	Licensing	ℹ	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	Licensing	⚠	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	Licensing	✗	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	
Audit	Log event	ℹ	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	No Action	<input type="checkbox"/>	

Page 17 of 19 [◀](#) [▶](#) [Apply](#)

4. Click **[Apply]**.
5. Go to **Overview > Server > Logs > Settings**.
6. Select the **Remote Syslog Enabled** checkbox.
7. Specify up to three Syslog server IP addresses, change the UDP port if required, and then click **[Apply]**.

The screenshot shows the 'Logs' tab selected in the top navigation bar. Below it, the 'Settings' tab is active. The main content area is titled 'Remote Syslog Settings'. It contains a form with the following fields:

- Remote Syslog Enabled:
- Syslog Server1:
- Syslog Server2:
- Syslog Server3:
- Port Number:

At the bottom right of the form is a 'Apply' button.

45.2. Configuring via the Command Line

1. Log in to iDRAC via SSH.
2. Run the following commands. Replace **ALERT**, **ACTION**, **NUMBER**, and **IP_ADDRESS** with the required values (see below).

```
# racadm eventfilters set -c ALERT -a ACTION -n NOTIFICATION
# racadm set iDRAC.Syslog.SyslogEnable 1
# racadm set iDRAC.Syslog.Server[NUMBER] IP_ADDRESS
```

- **ALERT:** the alert descriptor, in the format of `idrac.alert.category.[subcategory].[severity]`. Available categories are `all`, `system`, `storage`, `updates`, `audit`, `config`, and `worknotes`. Valid severity values are `critical`, `warning`, and `info`.
- **ACTION:** an action for this alert. Possible values are `none`, `powercycle`, `poweroff`, and `systemreset`.
- **NOTIFICATION:** required notifications for the alert. Valid values are `all` or `none`, or a comma-separated list including one or more of `snmp`, `ipmi`, `lcd`, `email`, and `remotesyslog`.
- **NUMBER:** the Syslog server number—`1`, `2` or `3`.
- **IP_ADDRESS:** the address of the NXLog agent.

Example 183. Configuring Syslog Logging to 192.168.6.143

The following commands disable all alert actions, enable Syslog notifications for all alerts (disabling other notifications), and enable Syslog logging to 192.168.6.143 (UDP port 514).

WARNING

This example disables any previously configured alert actions or notifications. Different `eventfilters` arguments must be used to enable or retain other action or notification types.

```
# racadm eventfilters set -c idrac.alert.all -a none -n remotesyslog  
# racadm set iDRAC.Syslog.SyslogEnable 1  
# racadm set iDRAC.Syslog.Server[1] 192.168.6.143
```

Chapter 46. Dell PowerVault MD Series

PowerVault MD logs can be sent to a remote Syslog destination via UDP by using the Event Monitor Windows service, which is a part of the Modular Disk Storage Manager application used to manage PowerVault. The MD Storage Manager is a separate application which is usually installed on a management server. It connects to the MD unit and provides a convenient graphical interface for managing the PowerVault storage.

Log Sample

```
Date/Time: 4/5/17 2:43:00 PM↵
Sequence number: 418209↵
Event type: 4011↵
Description: Virtual disk not on preferred path due to failover↵
Event specific codes: 0/0/0↵
Event category: Error↵
Component type: RAID Controller Module↵
Component location: Enclosure 0, Slot 0↵
Logged by: RAID Controller Module in slot 0↵
↵
Date/Time: 4/5/17 4:06:21 PM↵
Sequence number: 418233↵
Event type: 104↵
Description: Needs attention condition resolved↵
Event specific codes: 0/0/0↵
Event category: Internal↵
Component type: RAID Controller Module↵
Component location: Enclosure 0, Slot 0↵
Logged by: RAID Controller Module in slot 0
```

For more details about configuring PowerVault alerts and using MD Storage Manager, see [Dell Support](#).

NOTE

The steps below have been tested with the PowerVault MD3200 Series SAN and should work with any MD unit managed by MD Storage Manager Enterprise.

1. Configure NXLog for receiving log entries via UDP (see the [examples](#) below), then restart NXLog.
2. Confirm that the NXLog agent is accessible from the server where MD Storage Manager is installed.
3. Locate the **PMServer.properties** file. By default, the file can be found in **C:\Program Files (x86)\Dell\MD Storage Software\MD Storage Manager\client\data**.
4. Edit the file. Set **enable_local_logger** to **true**, specify the Syslog server address, and set the facility.

Example 184. Sending Logs to 192.168.15.223

With the following directives, the MD Storage Manager will send events to 192.168.15.223 via UDP port 514.

PMServer.properties

```
Time_format(12/24)=12
syslog_facility=3
DBM_files_maximum_key=20
DBM_files_minimum_key=5
syslog_receivers=192.168.15.223
DBM_recovery_interval_key=120
DBM_recovery_debounce_key=5
DBM_files_maintain_timeperiod_key=14
eventlog_source_name=StorageArray
enable_local_logger=true
syslog_tag=StorageArray
```

5. Restart the Event Monitor service to apply the changes.

Example 185. Receiving Syslog Messages From the MD Storage Manager

This example shows PowerVault logs as received and processed by NXLog.

nxlog.conf

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10  Module im_udp
11  Host 0.0.0.0
12  Port 514
13  Exec parse_syslog();
14 </Input>
15
16 <Output file>
17  Module om_file
18  File  "/var/log/mdsan.log"
19  Exec  to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.15.231",
  "EventReceivedTime": "2017-04-05 14:43:45",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 3,
  "SyslogFacility": "DAEMON",
  "SyslogSeverityValue": 4,
  "SyslogSeverity": "WARNING",
  "SeverityValue": 3,
  "Severity": "WARNING",
  "Hostname": "192.168.5.18",
  "EventTime": "2017-04-05 14:43:00",
  "SourceName": "StorageArray",
  "Message": "MD3620f1;4011;Warning;Virtual disk not on preferred path due to failover"
}
{
  "MessageSourceAddress": "192.168.15.231",
  "EventReceivedTime": "2017-04-05 16:07:01",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 3,
  "SyslogFacility": "DAEMON",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "192.168.5.18",
  "EventTime": "2017-04-05 16:06:21",
  "SourceName": "StorageArray",
  "Message": "MD3620f1;104;Informational;Needs attention condition resolved"
}
```

Example 186. Extracting Additional Fields

With the following configuration, NXLog will extract additional fields from the message.

nxlog.conf

```
1 <Input in_syslog_udp>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /^([a-zA-Z0-9]*);([0-9]*);([a-zA-Z]*);(.*)$/
8     {
9       $MDArray = $1;
10      $MDMsgID = $2;
11      $MDMsgLevel = $3;
12      $MDMessage = $4;
13    }
14  </Exec>
15 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.15.231",
  "EventReceivedTime": "2017-04-05 14:43:45",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 3,
  "SyslogFacility": "DAEMON",
  "SyslogSeverityValue": 4,
  "SyslogSeverity": "WARNING",
  "SeverityValue": 3,
  "Severity": "WARNING",
  "Hostname": "192.168.5.18",
  "EventTime": "2017-04-05 14:43:00",
  "SourceName": "StorageArray",
  "Message": "MD3620f1;4011;Warning;Virtual disk not on preferred path due to failover",
  "MDArray": "MD3620f1",
  "MDMsgID": "4011",
  "MDMsgLevel": "Warning",
  "MDMessage": "Virtual disk not on preferred path due to failover"
}
```

Chapter 47. DNS Monitoring

Monitoring and proactively analyzing DNS (Domain Name Server) queries and responses has become a standard security practice for networks of all sizes. Many types of malware rely on DNS traffic to communicate with *command and control* servers, inject ads, redirect traffic, or transport data masked as *payload* in DNS traffic to bypass intrusion detection systems.

Among other things, DNS traffic analysis is commonly used to:

- discover unknown devices that appear on the network;
- monitor critical devices that have not issued a query within a predefined time window;
- detect malware from young/esoteric domain lookups or consistent lookup failures; and
- analyze host, subnet, or user behavioral patterns.

TIP DNS traffic can quickly become overwhelming. To save resources the network administrator should drop fields that are not considered important.

According to [RFC 7626](#) there are no specific privacy laws for DNS data collection, in any country. However it is not clear if data protection directive 95/46/EC of the European Union includes DNS traffic collection.

47.1. Bind9 on Linux

Bind9 supports multiple logging configurations via *categories* and *channels*. Though Bind9 configuration is outside the scope of this document, the example below shows a standard configuration for writing all queries to a specific log file. A corresponding NXLog configuration parses the logs for processing.

Example 187. Bind9 Query Logging

The following Bind9 configuration will log all DNS queries to `/var/log/named/queries.log`.

`/etc/bind/named.conf`

```
logging {  
    channel query.log {  
        file "/var/log/named/queries.log";  
        print-severity yes;  
        print-time yes;  
        severity debug 3;  
    };  
  
    category queries { query.log; };  
};
```

Queries are then logged by Bind9 in a straightforward line-based format.

Log Sample

```
23-Mar-2017 06:38:30.142 info: client 192.168.100.105#58985 (_http._tcp.security.ubuntu.com):  
query: _http._tcp.security.ubuntu.com IN SRV + (192.168.100.105)  
23-Mar-2017 06:38:30.142 info: client 192.168.100.105#42195 (security.ubuntu.com): query:  
security.ubuntu.com IN A + (192.168.100.105)  
23-Mar-2017 06:38:30.143 info: client 192.168.100.105#42195 (security.ubuntu.com): query:  
security.ubuntu.com IN AAAA + (192.168.100.105)  
23-Mar-2017 06:38:30.143 info: client 192.168.100.105#34411 (_http._tcp.archive.ubuntu.com):  
query: _http._tcp.archive.ubuntu.com IN SRV + (192.168.100.105)
```

The following configuration uses a regular expression and named capture groups to extract the fields from each record. The [parsedate\(\)](#) function is used to convert the date and time strings to a [datetime](#) type value,

which is stored to `$EventTime`. Here, the output is written to file in JSON format; other output modules and formats provided by NXLog could be used instead.

nxlog.conf

```

1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 define REGEX_BIND_QUERIES /(?x)(?<Date>\d+-\w+-\d+)\s+ \
6   (?<Time>\d+:\d+:\d+\.\d+)\s+ \
7   (?<Severity>\w+)\s+ \
8   \w+\s+ \
9   (?<RemoteIP>(?:[0-9a-f]{0,4}:){1,7}[0-9a-f]{1,4}| \
10 (?:[0-9]{1,3}\.){3}[0-9]{1,3})#+\d+\s+ \
11  \((?:<QName>[^)]*)\)\s+ \
12  query:\s+\$+\s+\w+\s+ \
13  (?<QType>\w+)\s+ \
14  (?<RFlags>\+\w*)/
15
16 <Input dns_queries>
17   Module im_file
18   File "/var/log/named/queries.log"
19   <Exec>
20     if $raw_event =~ %REGEX_BIND_QUERIES%
21       $EventTime = parsedate($1 + " " + $2);
22   </Exec>
23 </Input>
24
25 <Output dns_out>
26   Module om_file
27   File "/tmp(nxlog-dns.json"
28   Exec to_json();
29 </Output>
```

Output Sample

```
{
  "EventReceivedTime": "2017-03-28 07:55:34",
  "SourceModuleName": "dns_queries",
  "SourceModuleType": "im_file",
  "Date": "28-Mar-2017",
  "QName": "design.com",
  "QType": "A",
  "RFlags": "+E",
  "RemoteIP": "127.0.0.1",
  "Severity": "info",
  "Time": "07:17:09.816",
  "EventTime": "2017-03-28 07:17:09"
}
```

47.2. Windows Server DNS

DNS logging can be set up on Windows Server by using either [ETW tracing](#) or [DNS Debug Logging](#).

47.2.1. Using ETW Trace Providers

DNS logs can also be captured by enabling DNS Analytical Logging and using Event Tracing for Windows (ETW). For more information, see [DNS Logging and Diagnostics](#) on Microsoft TechNet.

With *Analytical Logging* enabled, NXLog can use the `im_etw` module to collect DNS logs from the *Microsoft-Windows-DNSServer* ETW provider. This is the preferred method for collecting logs from Windows Server versions 2012 R2 and later, but this method is not available on earlier versions of Windows Server.

NOTE On Windows Server 2012 R2, this feature is provided by [hotfix 2956577](#).

Example 188. Using `im_etw`

The following configuration collects DNS logs via ETW and then writes the output to file in JSON format.

`nxlog.conf`

```

1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Input etw_in>
6   Module      im_etw
7   Provider    Microsoft-Windows-DNSServer
8 </Input>
9
10 <Output etw_out>
11  Module     om_file
12  File       'C:\etw_dns.json'
13  Exec       to_json();
14 </Output>
15
16 <Route r>
17  Path       etw_in => etw_out
18 </Route>
```

Output Sample

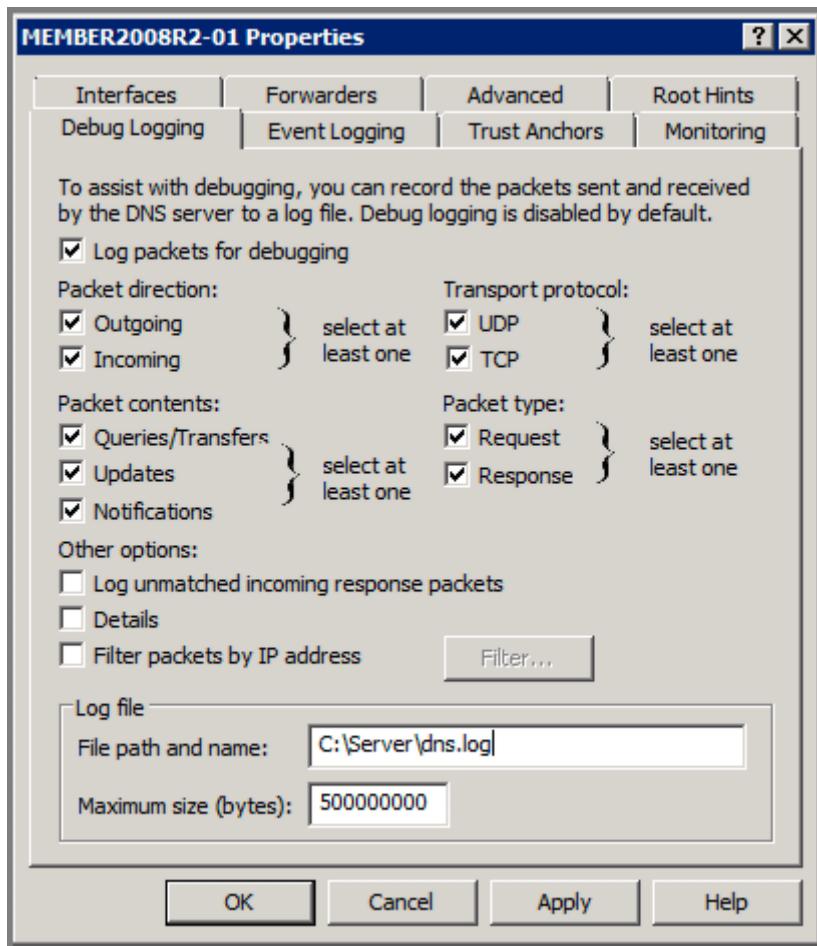
```
{
  "EventTime": "2017-03-10 09:51:03",
  "Provider": "Microsoft-Windows-DNSServer",
  "TCP": "0",
  "InterfaceIP": "10.2.0.162",
  "Source": "10.2.0.198",
  "RD": "1",
  "QNAME": "nickelfreesolutions.com.",
  "QTYPE": "1",
  "XID": "11675",
  "Port": "22416",
  "Flags": "256",
  "BufferSize": "41",
  "PacketData": "0x2D9B01000010000000000136E69636B656C66726565736F6C7574696F6E7303636F6D0000010001",
  "EventReceivedTime": "2017-03-10 09:51:04",
  "SourceModuleName": "etw_in",
  "SourceModuleType": "im_etw"
}
```

47.2.2. DNS Debug Logging

DNS logging can be enabled with *debug logging* mode. Queries are logged one per line.

To enable DNS debug logging, perform the following actions.

1. Open the **DNS Management** console (`dhsmgmt.msc`).
2. Right-click on the DNS server and choose **Properties** from the context menu.
3. Under the **Debug Logging** tab, enable **Log packets for debugging**.



4. Mark the check boxes corresponding to the data that should be logged.

WARNING

The **Details** option will produce multi-line logs. The following sections do not support parsing of this detailed format. However, NXLog could be configured to parse this format with the [xm_multiline](#) module and a regular expression.

5. Set the **File path and name** to the desired log file location.

WARNING

The Windows DNS service may not recreate the debug log file after a rollover. If you encounter this issue, be sure to use the C: drive for the debug log path. See the following post on the NXLog website: "[The disappearing Windows DNS debug log](#)".

Log Sample (Standard Debug Mode)

4/21/2017 7:52:03 AM 06B0 PACKET 0000000028657F0 UDP Snd 10.2.0.1	6590 R Q [8081 DR
NOERROR] A (7)example(3)com(0)↔	

See the following sections for information about parsing the logs.

47.2.2.1. Parsing With xm_msdns

The [xm_msdns](#) module, available in NXLog Enterprise Edition, can be used for parsing Windows Server DNS logs.

WARNING

This module does not support parsing of DNS Debugging logs generated with the [Details](#) option enabled.

NOTE

This module has been tested under Windows Server versions 2008 R2, 2012 R2, and 2016.

Example 189. Using xm_msdns

This configuration uses the [im_file](#) and [xm_msdns](#) modules to read and parse the log file. Output is written to file in JSON format for this example.

nxlog.conf

```

1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Extension dns_parser>
6   Module      xm_msdns
7   EventLine   TRUE
8   PacketLine  TRUE
9   NoteLine    TRUE
10 </Extension>
11
12 <Input in>
13   Module      im_file
14   File        'C:\Server\dns.log'
15   InputType   dns_parser
16 </Input>
17
18 <Output out>
19   Module      om_file
20   File        'C:\xm_dns.json'
21   Exec        to_json();
22 </Output>
```

Output Sample

```
{
  "EventTime": "2017-04-21 07:52:03",
  "ThreadId": "06B0",
  "Context": "PACKET",
  "InternalPacketIdentifier": "0000000028657F0",
  "Protocol": "UDP",
  "SendReceiveIndicator": "Snd",
  "RemoteIP": "10.2.0.1",
  "Xid": "6590",
  "QueryResponseIndicator": "Response",
  "Opcode": "Standard Query",
  "FlagsHex": "8081",
  "RecursionDesired": true,
  "RecursionAvailable": true,
  "ResponseCode": "NOERROR",
  "QuestionType": "A",
  "QuestionName": "example.com",
  "EventReceivedTime": "2017-04-21 7:52:03",
  "SourceModuleName": "in",
  "SourceModuleType": "im_file"
}
```

47.2.2.2. Parsing With Regular Expressions

While the `xm_msdns` module is the preferred method for parsing DNS logs, and is about three times faster, regular expressions can also be used.

Example 190. Parsing DNS Logs With Regular Expressions

This example parses the DNS Debugging logs from file and then writes the output to file in JSON format.

WARNING This configuration does not parse DNS Debugging logs generated with the [Details](#) option enabled.

NOTE This has been tested under Windows Server versions 2008 R2, 2012 R2, and 2016.

`nxlog.conf`

```
1 define EVENT_REGEX /(?x)(?<Date>\d+(?:\.\d+){2})\s \
2                               (?<Time>\d+(?:\.\d+){2}\s+w+)\s \
3                               (?<ThreadId>w+)\s+ \
4                               (?<Context>w+)\s+ \
5                               (?<InternalPacketIdentifier>[[ :xdigit:]])+\s+ \
6                               (?<Protocol>w+)\s+ \
7                               (?<SendReceiveIndicator>w+)\s \
8                               (?<RemoteIP>[[ :xdigit:]])+\s+ \
9                               (?<Xid>[[ :xdigit:]])+\s \
10                          (?<QueryType>\s|R)\s \
11                          (?<Opcode>[A-Z]|\?)\s \
12                          (?<QFlags>\[(.*?)\])\s+ \
13                          (?<QuestionType>w)\s+ \
14                          (?<QuestionName>.*)/
15 define EMPTY_EVENT_REGEX /(^$|^\s+$)/
16 define DOMAIN_REGEX /\(\d+\)([\w-]+)\(\d+\)([\w-]+)/
17 define SUBDOMAIN_REGEX /\(\d+\)([\w-]+)\(\d+\)([\w-]+)\(\d+\)(\w+)/
18 define NOT_STARTING_WITH_DATE_REGEX /^(!\d+/\d+/\d+).*/
19 define QFLAGS_REGEX /(?x)(?<FlagsHex>\d+)\s+ \
20                               (?<FlagsCharCodes>\s+|[A-Z]{2}|[A-Z]))\s+ \
21                               (?<ResponseCode>w+)/
22
23 <Extension _json>
24     Module xm_json
25 </Extension>
26
27 <Input in>
28     Module im_file
29     File    'C:\Server\dns.log'
30     <Exec>
31         # Drop entries that have empty lines
32         if $raw_event =~ %EMPTY_EVENT_REGEX% drop();
33         # Drop entries not starting with date
34         if $raw_event =~ %NOT_STARTING_WITH_DATE_REGEX% drop();
35         # Split entries into fields & define regular entries
36         if $raw_event =~ %EVENT_REGEX%
37         {
38             $Regular = TRUE;
39             $EventTime = parsedate($Date + " " + $Time);
40             $Raw = $raw_event;
41             delete($date);
42             delete($time);
43             if $FlagsCharCodes =~ /^$ / delete($FlagsCharCodes );
```

```

44      # Convert domains from (8)mydomain(1)com to mydomain.com
45      if $QuestionName =~ %DOMAIN_REGEX% $QuestionName = $1 + "." + $2;
46      # Convert domains from (8)sub(8)mydomain(1)com to sub.mydomain.com
47      if $QuestionName =~ %SUBDOMAIN_REGEX%
48          $QuestionName = $1 + "." + $2 + "." + $3;
49
50      # Set query flags
51      if $QFlags =~ %QFLAGS_REGEX% delete($QFlags);
52
53      # Set the query type
54      if $QueryType =~ %EMPTY_EVENT_REGEX% $QueryType = "query";
55      else $QueryType = "response";
56  }
57  else
58  {
59      $Regular = FALSE;
60      $Raw = $raw_event;
61  }
62  </Exec>
63 </Input>
64
65 <Output out>
66   Module  om_file
67   Exec    to_json();
68   File    'C:\output-dns-traffic.json'
69 </Output>

```

Output Sample

```
{
  "EventReceivedTime": "2017-04-21 07:52:16",
  "SourceModuleName": "in",
  "SourceModuleType": "im_file",
  "Context": "PACKET",
  "InternalPacketIdentifier": "0000000028657F0",
  "Opcode": "Q",
  "Protocol": "UDP",
  "QueryType": "response",
  "QuestionName": "notabilus.com",
  "QuestionType": "A",
  "RemoteIP": "10.2.0.1",
  "SendReceiveIndicator": "Snd",
  "ThreadId": "06B0",
  "Xid": "6590",
  "Regular": true,
  "EventTime": "2017-04-21 07:52:03",
  "Raw": "4/21/2017 7:52:03 AM 06B0 PACKET 0000000028657F0 UDP Snd 10.2.0.1      6590 R Q
[8081 DR NOERROR] A      (9)notabilus(3)com(0)",
  "FlagsCharCodes": "DR",
  "FlagsHex": "8081",
  "ResponseCode": "NOERROR"
}
```

Chapter 48. Elasticsearch and Kibana

Elasticsearch is a search engine and document database that is commonly used to store logging data. Kibana is a popular user interface and querying front-end for Elasticsearch. Kibana is often used with the Logstash data collection engine—together forming the *ELK* stack (Elasticsearch, Logstash, and Kibana).

However, Logstash is not actually required to load data into Elasticsearch. NXLog can do this as well, and offers several advantages over Logstash—this is the *KEN* stack (Kibana, Elasticsearch, and NXLog).

- Because Logstash is written in Ruby and requires Java, it has high system resource requirements. NXLog has a small resource footprint and is recommended by many ELK users as the log collector of choice for Windows and Linux.
- Due to the Java dependency, Logstash requires system administrators to deploy the Java runtime onto their production servers and keep up with Java security updates. NXLog does not require Java.
- The EventLog plugin in Logstash uses the Windows WMI interface to retrieve the EventLog data. This method incurs a significant performance penalty. NXLog uses the Windows EventLog API natively in order to efficiently collect EventLog data.

The following sections provide details for configuring NXLog to:

- [send logs directly to Elasticsearch](#), replacing Logstash; or
- [forward collected logs to Logstash](#), acting as a log collector for Logstash.

48.1. Sending Logs to Elasticsearch

It is assumed that the Elasticsearch and Kibana servers are already installed. Consult the [Elasticsearch Reference](#) and the [Kibana User Guide](#) for more information.

1. For Kibana's time filters to work correctly, an index template must be defined in Elasticsearch. The following command will push a template to Elasticsearch indicating that the `EventTime` field should be parsed as a date in the given format. This will be applied to all records in indices beginning with `nxlog`.

```
$ curl -XPUT 'localhost:9200/_template/nxlog?pretty' \
  -H 'Content-Type: application/json' -d'
{ "template": "nxlog*", \
  "mappings": { "_default_": { "properties": { "EventTime": { \
    "type": "date", \
    "format": "YYYY-MM-dd HH:mm:ss" } } } } }
```

2. Configure NXLog.

Example 191. Using om_elasticsearch

The [om_elasticsearch](#) module is only available in NXLog Enterprise Edition. Because it sends data in batches, it reduces the effect of the latency inherent in HTTP responses, allowing the Elasticsearch server to process the data much more quickly (10,000 EPS or more on low-end hardware).

nxlog.conf

```

1 <Output out>
2   Module      om_elasticsearch
3   URL        http://localhost:9200/_bulk
4   FlushInterval 2
5   FlushLimit 100
6
7   # Create an index daily
8   Index       strftime($EventTime, "nxlog-%Y%m%d")
9   IndexType   "My logs"
10
11  # Use the following if you do not have $EventTime set
12  #Index      strftime($EventReceivedTime, "nxlog-%Y%m%d")
13 </Output>
```

NOTE

The [IndexType](#) parameter is not mandatory, but it helps with sorting log entries on the Kibana dashboard. The parameter expects a string expression, and is evaluated for each event record.

Example 192. Using om_http

For NXLog Community Edition, the [om_http](#) module can be used instead to send logs to Elasticsearch. Because it sends a request to the Elasticsearch HTTP REST API for each event, the maximum logging throughput is limited by HTTP request and response latency. Therefore this method is suitable only for low-volume logging scenarios.

nxlog.conf

```

1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Output out>
6   Module      om_http
7   URL        http://localhost:9200
8   ContentType application/json
9   <Exec>
10     set_http_request_path(strftime($EventTime, "/nxlog-%Y%m%d/" +
11                               $SourceModuleName));
12     rename_field("timestamp", "@timestamp");
13     to_json();
14   </Exec>
15 </Output>
```

3. Restart NXLog.
4. Configure the appropriate index pattern for Kibana.
 - a. Go to the **Settings > Indices** tab.
 - b. Enable the **Index contains time-based events** check box.
 - c. Disable the **Use event times to create index names** check box.

d. Set the **Index name or pattern** to **nxlog-***.

Until you start sending logs into Elasticsearch, you will not be able to set the index pattern and will receive the error *Unable to fetch mapping. Do you have indices matching the pattern?*

NOTE

The following command can be used to verify that logs are being received by Elasticsearch. You should see at least one **nxlog-*** index and the **docs.count** counter increasing as logs arrive.

```
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
```

e. Set the **Time-field name** selector to **EventTime**.

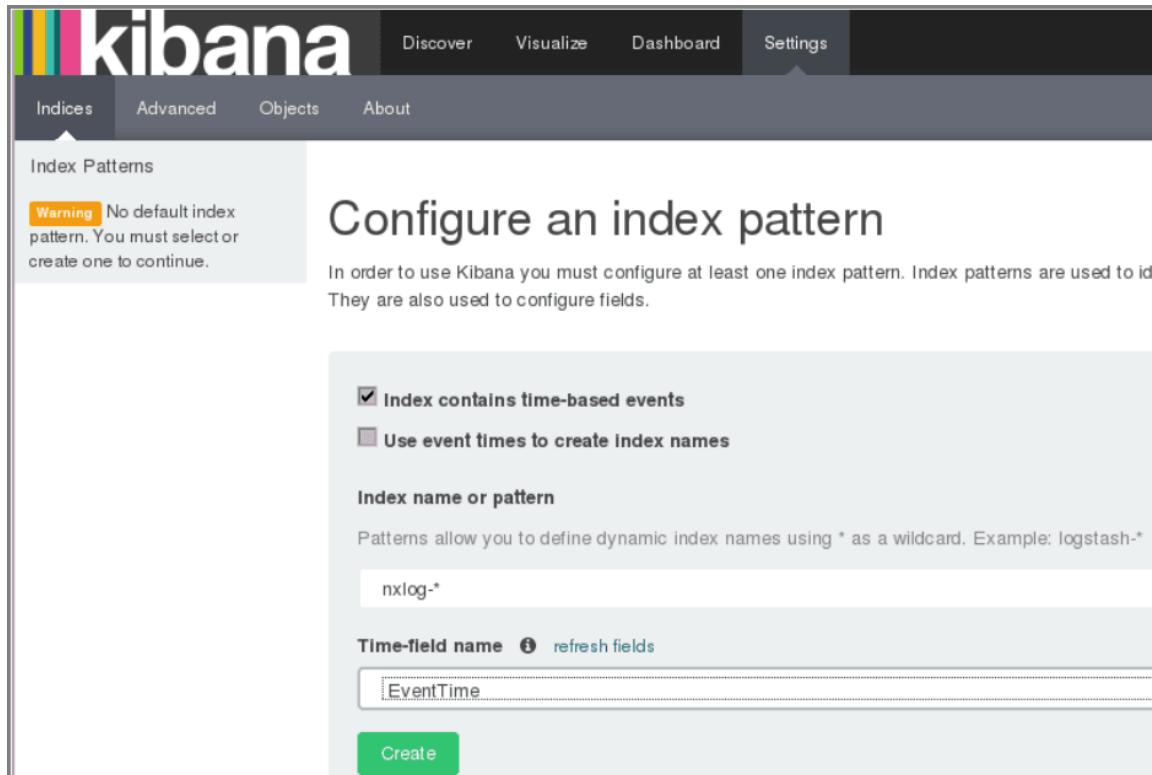


Figure 2. Kibana Index Pattern

5. Test that your NXLog and Elasticsearch/Kibana configuration is working.

- Make sure that the client logs or event sources are sending data.
- Visit the Kibana web interface. You should see events identified with **nxlog-*** keys.

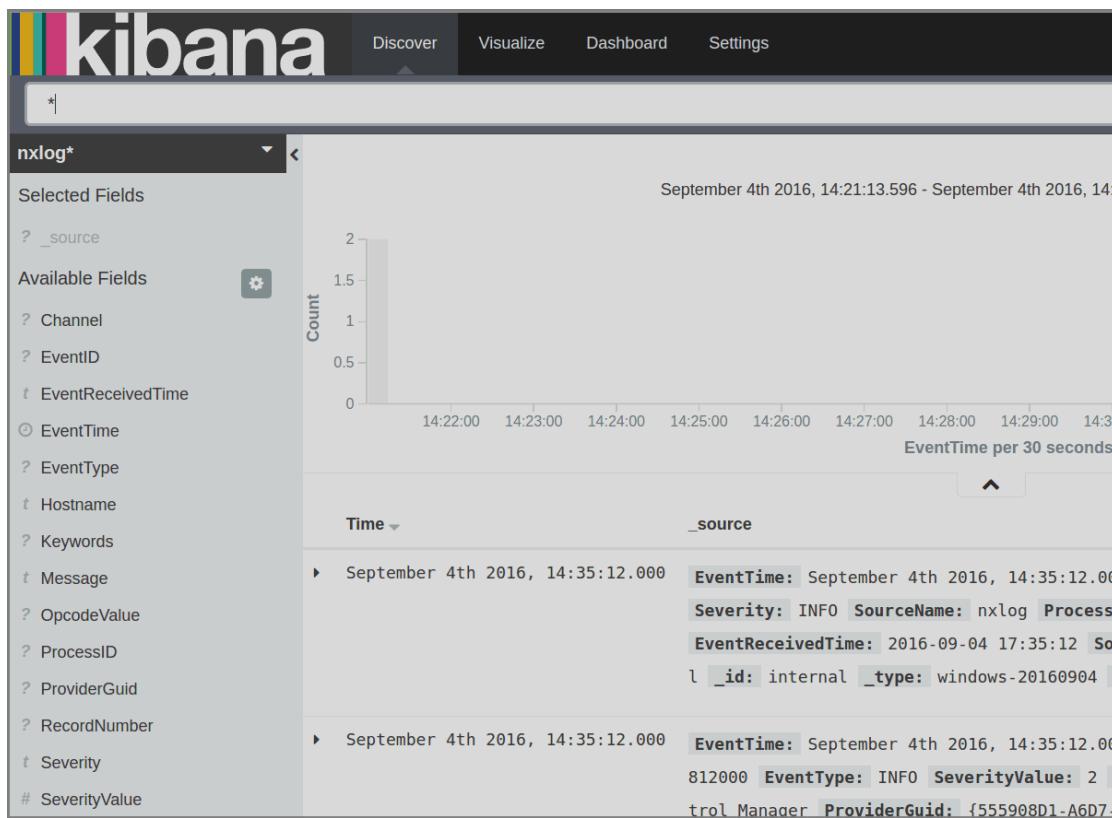


Figure 3. Kibana Events

48.2. Forwarding Logs to Logstash

NXLog can be configured to act as a log collector, forwarding logs to Logstash in JSON format.

1. Set up a configuration on the Logstash server to process incoming event data from NXLog.

logstash.conf

```

input {
  tcp {
    codec => json_lines { charset => CP1252 }
    port => "3515"
    tags => [ "tcpjson" ]
  }
}

filter {
  date {
    locale => "en"
    timezone => "Etc/GMT"
    match => [ "EventTime", "YYYY-MM-dd HH:mm:ss" ]
  }
}

output {
  elasticsearch {
    host => localhost
  }
  stdout { codec => rubydebug }
}

```

The `json` codec in Logstash sometimes fails to properly parse JSON—it will concatenate more than one JSON record into one event. Use the `json_lines` codec instead.

NOTE

Although the `im_msvisalog` module converts data to UTF-8, Logstash seems to have trouble parsing that data. The `charset => CP1252` seems to help.

2. Configure NXLog.

nxlog.conf

```
1 <Output out>
2   Module  om_tcp
3   Host    10.1.1.1
4   Port    3515
5   Exec    to_json();
6 </Output>
```

3. Restart NXLog.

Chapter 49. F5 BIG-IP

F5 BIG-IP appliances are capable of sending their logs to a remote Syslog destination via TCP or UDP. When sending logs over the network, it is recommended to use TCP as the more reliable protocol. With UDP there is a potential to lose entries, especially when there is a high volume of messages.

There are multiple sub-systems that write logs to different files. Below is an example of Local Traffic Management (LTM) logs reporting pool members being up or down.

Local Traffic Management (LTM) Log Sample

```
Mar 14 16:50:12 1-lb1 notice mcpd[7660]: 01070639:5: Pool /Common/q-qa-pool member /Common/q-qa1:25  
session status forced disabled.  
Mar 14 16:51:33 1-lb1 notice mcpd[7660]: 01070639:5: Pool /Common/q-qa-pool member /Common/q-qa1:25  
session status enabled.  

```

The following audit logs are written to a different local file.

Audit Log Sample

```
Mar 14 16:43:41 1-lb1 notice httpd[3064]: 01070417:5: AUDIT - user john - RAW: httpd(mod_auth_pam):  
user=john(john) partition=[All] level=Administrator tty=/usr/bin/tmsh host=192.168.9.78 attempts=1  
start="Tue Mar 14 16:43:41 2017".  
Mar 14 17:10:33 1-lb1 notice httpd[1181]: 01070417:5: AUDIT - user john - RAW: httpd(mod_auth_pam):  
user=john(john) partition=[All] level=Administrator tty=/usr/bin/tmsh host=192.168.9.78 attempts=1  
start="Tue Mar 14 16:43:41 2017" end="Tue Mar 14 17:10:33 2017".  

```

For more details on BIG-IP log files and how to view them, please refer to the [K16197](#) knowledge base article. Additional information on configuring logging options on BIG-IP devices can be found in the [F5 Knowledge Center](#). Select the appropriate software version and look for the "Log Files" section in the TMOS Operations Guide.

NOTE

The steps below have been tested with BIG-IP v11 but should also work for other versions.

49.1. Collecting BIG-IP Logs via TCP

The BIG-IP web interface does not provide a way to configure an external TCP Syslog destination, so this must be done via the command line.

1. Configure NXLog to receive log entries via TCP and process them as Syslog (see the [examples](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from all BIG-IP devices being configured. A new route or default gateway may need to be configured depending on the current setup.
3. Connect via SSH to the BIG-IP device. In case of a High Availability (HA) group, make sure you are logged into the active unit. You should see **(Active)** in the command prompt.
4. Review the existing Syslog configuration on BIG-IP and remove it.

```
# tmsh list sys syslog include  
# tmsh modify sys syslog include none
```

5. Configure a remote Syslog destination on BIG-IP. Replace **IP_SYSLOG** and **PORT** with the IP address and port that the NXLog agent is listening on. Replace **LEVEL** with the required logging level.

```
# tmsh modify sys syslog include "destination remote_server \  
{tcp(\"IP_SYSLOG\" port (PORT));};filter f_alllogs \  
{level (LEVEL...emerg);};log {source(local);filter(f_alllogs);\  
destination(remote_server);};"
```

NOTE

This command forwards all appliance logs to the remote destination, so nothing will be logged locally as soon as it is executed.

Example 193. Redirecting Informational Logs via TCP

This command redirects logs at the informational level (from `info` to `emerg`) to an NXLog agent at 192.168.6.43, via TCP port 1514.

```
# tmsh modify /sys syslog include "destination remote_server \
{tcp(\"192.168.6.143\") port (1514)};filter f_alllogs \
{level (info...emerg)};log {source(local);filter(f_alllogs); \
destination(remote_server)};"
```

6. In case of a High Availability (HA) group, synchronize the configuration changes to the other units.

```
# tmsh run cm config-sync to-group GROUP_NAME
```

NOTE

Once the configuration has been synchronized to all members of the group, each member will be sending logs, inserting its own hostname and IP address. In the event of failover, logging will continue from both units regardless of which one is currently active.

Example 194. Receiving BIG-IP Logs via TCP

This configuration uses the `im_tcp` module to collect the BIG-IP logs. A JSON output sample shows the resulting logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module xm_json
7 </Extension>
8
9 <Input in>
10  Module im_tcp
11  Host 0.0.0.0
12  Port 1514
13  Exec parse_syslog();
14 </Input>
15
16 <Output out>
17  Module om_file
18  File  "/var/log/f5.log"
19  Exec to_json();
20 </Output>
```

Output Sample

```
{  
    "MessageSourceAddress": "192.168.6.161",  
    "EventReceivedTime": "2017-03-14 17:03:16",  
    "SourceModuleName": "in",  
    "SourceModuleType": "im_tcp",  
    "SyslogFacilityValue": 16,  
    "SyslogFacility": "LOCAL0",  
    "SyslogSeverityValue": 5,  
    "SyslogSeverity": "NOTICE",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "Hostname": "l-lb2",  
    "EventTime": "2017-03-14 17:03:53",  
    "SourceName": "mcpd",  
    "ProcessID": "7233",  
    "Message": "notice httpd[5150]: 01070639:5: Pool /Common/q-qa-pool member /Common/q-qa1:25  
session status enabled."  
}  
  
{  
    "MessageSourceAddress": "192.168.6.91",  
    "EventReceivedTime": "2017-03-14 17:10:18",  
    "SourceModuleName": "in",  
    "SourceModuleType": "im_tcp",  
    "SyslogFacilityValue": 16,  
    "SyslogFacility": "LOCAL0",  
    "SyslogSeverityValue": 5,  
    "SyslogSeverity": "NOTICE",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "Hostname": "l-lb1",  
    "EventTime": "2017-03-14 17:10:33",  
    "SourceName": "httpd",  
    "ProcessID": "1181",  
    "Message": "notice httpd[5150]: 01070417:5: AUDIT - user john - RAW: httpd(mod_auth_pam):  
user=john(john) partition=[All] level=Administrator tty=/usr/bin/tmsh host=192.168.9.78  
attempts=1 start=\"Tue Mar 14 16:43:41 2017\" end=\"Tue Mar 14 17:10:33 2017\"."  
}
```

NXLog can also be configured to extract additional fields from the messages, including those that contain key-value pairs.

Example 195. Extracting Fields From the BIG-IP Logs

This configuration uses the [xm_kvp](#) module to extract additional fields from the log messages.

nxlog.conf

```
1 <Extension _syslog>
2     Module      xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module      xm_json
7 </Extension>
8
9 <Extension kvp>
10    Module     xm_kvp
11    KVPDelimiter  " "
12    KVDelimiter    =
13    EscapeChar   \\
14 </Extension>
15
16 <Input in>
17     Module      im_tcp
18     Host        0.0.0.0
19     Port        1514
20     <Exec>
21         parse_syslog();
22         if $Message =~ /^([a-z]* ([a-zA-Z]*)(.*))$/
23         {
24             $F5MsgLevel = $1;
25             $F5Proc = $2;
26             $F5Message = $3;
27             if $F5Message =~ /^[0-9]*\]: ([0-9]*):([0-9]+): (.*)$/
28             {
29                 $F5MsgID = $1;
30                 $F5MsgSev = $2;
31                 $F5Msg = $3;
32                 if $F5Msg =~ /RAW: ([a-z]*)\(((a-zA-Z*))\): ([a-zA-Z]+=.+)/
33                 {
34                     $F5Process = $1;
35                     $F5Module = $2;
36                     kvp->parse_kvp($3);
37                 }
38             }
39         }
40     </Exec>
41 </Input>
42
43 <Output out>
44     Module      om_file
45     File        "/var/log/f5.log"
46     Exec        to_json();
47 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.6.91",
  "EventReceivedTime": "2017-04-16 00:06:43",
  "SourceModuleName": "in",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 10,
  "SyslogFacility": "AUTHPRIV",
  "SyslogSeverityValue": 5,
  "SyslogSeverity": "NOTICE",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "l-lb1",
  "EventTime": "2017-04-16 00:07:59",
  "Message": "notice httpd[5320]: 01070417:5: AUDIT - user john - RAW: httpd(mod_auth_pam): user=john(john) partition=[All] level=Administrator tty=/usr/bin/tmsh host=192.168.9.78 attempts=1 start=\"Sun Apr 16 00:07:59 2017\".",
  "F5MsgLevel": "notice",
  "F5Proc": "httpd",
  "F5Message": "[5320]: 01070417:5: AUDIT - user john - RAW: httpd(mod_auth_pam): user=john(john) partition=[All] level=Administrator tty=/usr/bin/tmsh host=192.168.9.78 attempts=1 start=\"Sun Apr 16 00:07:59 2017\".",
  "F5MsgID": "01070417",
  "F5MsgSev": "5",
  "F5Msg": "AUDIT - user john - RAW: httpd(mod_auth_pam): user=john(john) partition=[All] level=Administrator tty=/usr/bin/tmsh host=192.168.9.78 attempts=1 start=\"Sun Apr 16 00:07:59 2017\".",
  "F5Process": "httpd",
  "F5Module": "mod_auth_pam",
  "user": "john(john)",
  "partition": "[All]",
  "level": "Administrator",
  "tty": "/usr/bin/tmsh",
  "host": "192.168.9.78",
  "attempts": "1",
  "start": "Sun Apr 16 00:19:55 2017"
}
```

49.2. Collecting BIG-IP Logs via UDP

When reliable delivery is not a concern, or in case there is a requirement to have local copies of log entries on each appliance, BIG-IP logs can be sent to a remote Syslog destination via UDP.

1. Configure NXLog to receive log entries via UDP and process them as Syslog (see the [example](#) below). Then restart the agent.
2. Make sure the NXLog agent is accessible from all BIG-IP devices being configured. A new route or default gateway may need to be configured, depending on the current setup.
3. Proceed with the Syslog configuration on BIG-IP, using either the command line or the web interface. See the following sections.

Example 196. Receiving BIG-IP Logs via UDP

This configuration uses the **im_udp** module to collect the BIG-IP logs.

nxlog.conf

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input in_syslog_udp>
6   Module im_udp
7   Host 0.0.0.0
8   Port 514
9   Exec parse_syslog();
10 </Input>
```

49.2.1. Configuring via the Command Line

1. Connect via SSH to the BIG-IP device. In case of a High Availability (HA) group, make sure you are logged into the active unit. You should see (**Active**) in command prompt.
2. Configure a remote Syslog destination on BIG-IP. Replace **IP_SYSLOG** and **PORT** with the IP address and port that the NXLog agent is listening on.

```
# tmsh modify sys syslog remote-servers add { nxlog { \
host IP_SYSLOG remote-port PORT } }
```

Example 197. Redirecting Informational Logs via UDP

This command redirects Informational Logs to an NXLog agent at 192.168.6.143, via UDP port 514.

```
# tmsh modify sys syslog remote-servers add { nxlog { \
host 192.168.6.143 remote-port 514 } }
```

3. In case of a High Availability (HA) group, synchronize configuration changes to the other units:

```
# tmsh run cm config-sync to-group GROUP_NAME
```

NOTE

Once the configuration has been synchronized to all members of the group, each member will be sending logs, inserting its own hostname and IP address. In the event of failover, logging will continue from both units regardless of which one is currently active.

49.2.2. Configuring via the Web Interface

1. Log in to the BIG-IP web interface. In case of a High Availability (HA) group, make sure you are logged into the active unit. You should see **ONLINE (Active)** in the top left corner.
2. Go to **System > Logs > Configuration > Remote Logging**.
3. Type in the **Remote IP** and **Remote Port**, then click [**Add**] and [**Update**].

The screenshot shows the F5 BIG-IP configuration interface. The top status bar indicates 'Firewall: Consistent', 'ONLINE (ACTIVE)', 'In Sync', and 'Managed by BIG-IQ'. The left sidebar has a 'Main' tab selected, followed by 'Help' and 'About'. Under 'Main', there are links for 'Statistics', 'iApps', 'DNS', 'Local Traffic', 'Traffic Intelligence', 'Acceleration', 'Device Management', 'Security', 'Network', and 'System' (with 'Configuration' and 'Device Certificates' dropdowns). The main content area is titled 'System > Logs : Configuration : Remote Logging'. It includes tabs for 'System', 'Captured Transactions', 'Packet Filter', 'Local Traffic', and 'GSLB'. A sub-menu under 'Audit' shows 'Application Security', 'Audit', and 'Configuration' (which is highlighted in yellow). The 'Properties' section contains fields for 'Remote IP' (192.168.6.143), 'Remote Port' (514), and 'Local IP' (optional). An 'Add' button is available. Below this is a 'Remote Syslog Server List' table with columns for 'Edit' and 'Delete'. At the bottom of the main content area is an 'Update' button.

4. In case of a High Availability (HA) group, synchronize the configuration changes to the other units:
 - a. Click on the yellow **Changes Pending** in the top left corner.
 - b. Select Active unit which should be marked as **(Self)**.
 - c. Make sure **Sync Device to Group** option is chosen and click **[Sync]**.

The screenshot shows the F5 BIG-IP Device Management Overview page. At the top, there's a banner with the F5 logo and status indicators: Firewall: Consistent, ONLINE (ACTIVE), Changes Pending, and Managed by BIG-IQ. The main navigation bar includes Main, Help, and About. On the left, a sidebar lists various management sections like Statistics, iApps, DNS, Local Traffic, Traffic Intelligence, Acceleration, Device Management (with Overview selected), Security, Network, and System. The central panel is titled 'Device Management > Overview' and contains a 'Device Groups' table and a 'Sync Summary' section. The 'Device Groups' table lists four groups: Sync-failover, SSLs, datasync-global-dg, and device_trust_group, each with its sync status, number of devices, device group type, and sync type. Below this is a 'Sync Summary' box with tabs for Status, Changes Pending, Summary, and Details. The 'Devices' section shows two units, I-lb1 (Self) and I-lb2, with their HA status, names, sync status, and configuration times. A 'Sync Options' section includes radio buttons for Sync Device to Group (selected) and Sync Group to Device, and a checkbox for Overwrite Configuration. A 'Sync' button is at the bottom.

NOTE

Once the configuration has been synchronized to all members of the group, each member will be sending logs, inserting its own hostname and IP address. In the event of failover, logging will continue from both units regardless of which one is currently active.

49.3. Using SNMP Traps

BIG-IP devices are also capable of sending SNMP traps. The device contains predefined default SNMP traps which can be enabled during SNMP configuration. There is also an option to create user-defined traps. More information about SNMP support on BIG-IP devices can be found in the [F5 Knowledge Center](#) under the "Alerts" section in the TMOS Operations Guide.

BIG-IP systems also come with Management Information Base (MIB) files stored on the device itself. Additional information on that is available in [K13322](#).

1. Configure NXLog with the `xm_snmp` module. See the [example](#) below.
2. Make sure the NXLog agent is accessible from all BIG-IP devices being configured. A new route or default gateway may need to be configured, depending on the current setup.
3. Proceed with the SNMP configuration on BIG-IP, using either the command line or the web interface. See the following sections.

Example 198. Receiving SNMP Traps

This example NXLog configuration uses the `im_udp` and `xm_snmp` modules to receive SNMP traps. The corresponding JSON-formatted output is shown below.

nxlog.conf

```

1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Extension snmp>
6   Module      xm_snmp
7   MIBDir     /usr/share/mibs/bigip
8   # The following <User> section is required for SNMPv3
9   #<User snmp_user>
10  #   AuthProto sha1
11  #   AuthPasswd q1w2e3r4
12  #   EncryptPasswd q1w2e3r4
13  #   EncryptProto aes
14  #</User>
15 </Extension>
16
17 <Input in>
18   Module      im_udp
19   Host        0.0.0.0
20   Port        162
21   InputType   snmp
22 </Input>
23
24 <Output out>
25   Module      om_file
26   File        "/var/log/f5.log"
27   Exec        to_json();
28 </Output>
```

Output Sample

```
{
  "SNMP.CommunityString": "nxlog",
  "SNMP.RequestID": 449377444,
  "EventTime": "2017-03-18 16:37:41",
  "SeverityValue": 2,
  "Severity": "INFO",
  "OID.1.3.6.1.2.1.1.3.0": 1277437018,
  "OID.1.3.6.1.6.3.1.1.4.1.0": "1.3.6.1.4.1.3375.2.4.0.3",
  "OID.1.3.6.1.6.3.1.1.4.3.0": "1.3.6.1.4.1.3375.2.4",
  "MessageSourceAddress": "192.168.6.91",
  "EventReceivedTime": "2017-03-18 16:37:41",
  "SourceModuleName": "in",
  "SourceModuleType": "im_udp"
}
```

49.3.1. Configuring via the Command Line

1. Connect via SSH to the BIG-IP device. In case of a High Availability (HA) group, make sure you are logged into the active unit. You should see (**Active**) in the command prompt.
2. Enable the pre-defined default traps as required.

```
# tmsh modify sys snmp bigip-traps enabled
# tmsh modify sys snmp agent-trap enabled
# tmsh modify sys snmp auth-trap enabled
```

3. Create an SNMP user (SNMPv3 only).

```
# tmsh modify sys snmp users add { \
USERNAME { \
username USERNAME \
auth-protocol sha \
privacy-protocol aes \
auth-password *** \
privacy-password *** } }
```

Example 199. User snmp_user configured for MD5 and AES

```
# tmsh modify sys snmp users add { \
snmpv3_user { \
username snmpv3_user \
auth-protocol md5 \
privacy-protocol aes \
auth-password q1w2e3r4 \
privacy-password q1w2e3r4 } }
```

4. Configure the remote SNMP destination on BIG-IP. Replace **NAME**, **COMMUNITY**, **IP_ADDRESS**, and **PORT** with appropriate values. Replace **NETWORK** with **other** unless traps are going out the management interface, when **management** should be specified instead.

```
# tmsh modify sys snmp traps add { NAME { community COMMUNITY \
host IP_ADDRESS port PORT network NETWORK } }
```

Example 200. Sending Traps via SNMPv2

This command enables sending SNMPv2 traps to 192.168.6.143.

```
# tmsh modify sys snmp traps add { 192_168_6_143 { community nxlog \
host 192.168.6.143 port 162 network other } }
```

In case of SNMPv3, this command needs additional parameters, including security-level, auth-protocol, auth-password, privacy-protocol, and privacy-password.

Example 201. Sending Traps via SNMPv3

This command enables sending SNMPv3 traps to 192.168.6.143, using SHA and AES.

```
# tmsh modify sys snmp traps add { nxlog { \
version 3 \
host 192.168.6.143 \
port 162 \
network other \
security-level auth-privacy \
security-name snmp_user \
auth-protocol sha \
auth-password q1w2e3r4 \
privacy-protocol aes \
privacy-password q1w2e3r4 } }
```

NOTE

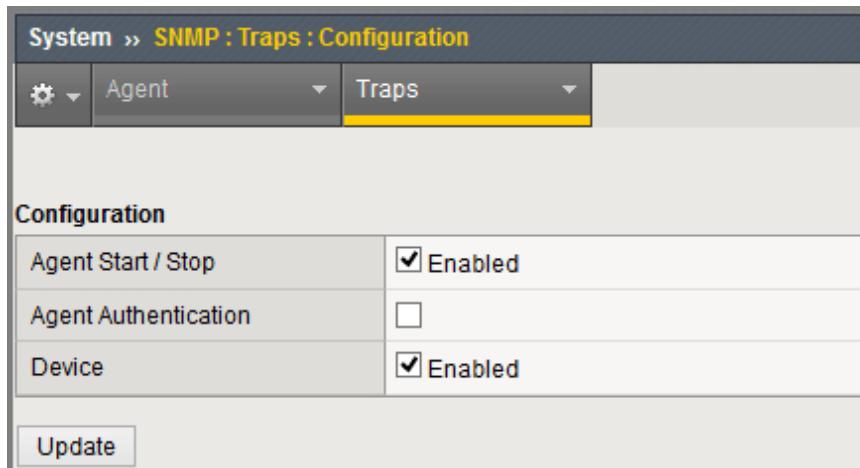
If the BIG-IP configuration has been previously migrated or cloned, SNMPv3 may not work because the EnginID is not unique. In this case it must be reset as described in [K6821](#).

5. In case of a High Availability (HA) group, synchronize the configuration changes to the other units.

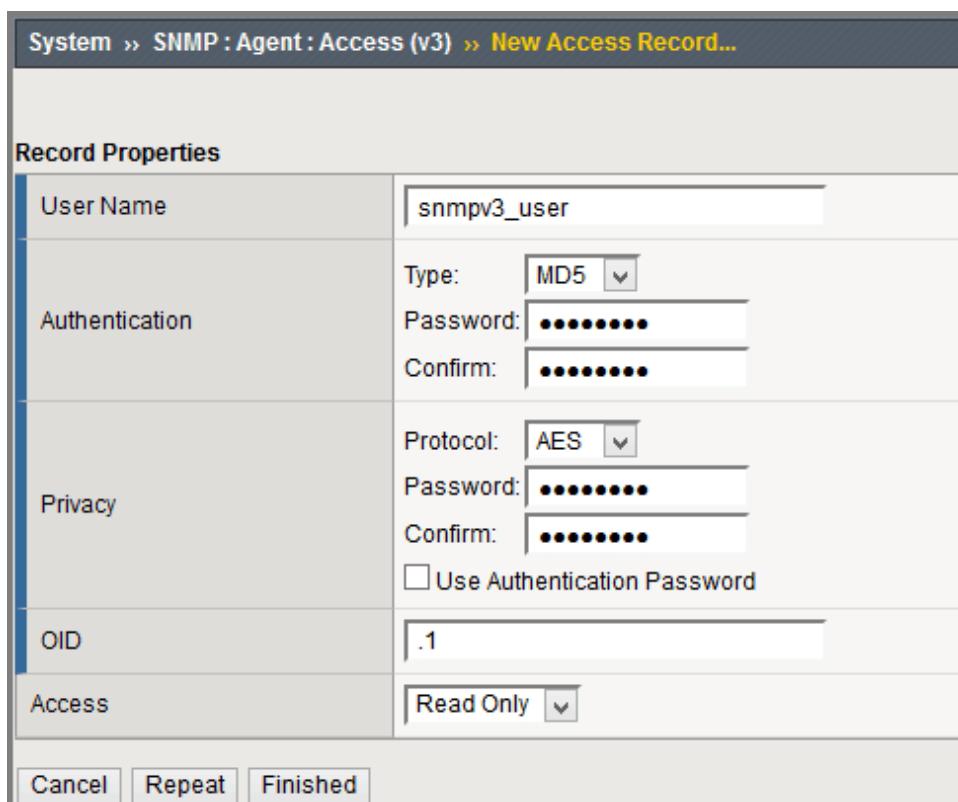
```
# tmsh run cm config-sync to-group GROUP_NAME
```

49.3.2. Configuring via the Web Interface

1. Log in to the BIG-IP web interface. In case of a High Availability (HA) group, make sure you are logged into the active unit. You should see **ONLINE (Active)** in the top left corner.
2. Go to **System > SNMP > Traps**. Select the required SNMP events and click [**Update**].



3. Create an SNMP user (SNMPv3 only). Go to **System > SNMP > Agent > Access (v3)**. Click [**Create**] and specify the user name, authentication type and password, privacy protocol and password, and access type. Specify an OID value to limit access to certain OIDs, or use **.1** to allow full access.



4. Go to **System > SNMP > Traps > Destination** and click [**Create**]. Specify the SNMP version, community

name, destination IP address, destination port, and network to send traffic to. Then click [**Finished**].

System » SNMP : Traps : Destination » New Trap Record...

Record Properties	
Version	v2c
Community	nxlog
Destination	192.168.6.143
Port	161
Network	Default

Cancel **Repeat** **Finished**

SNMPv3 requires additional parameters. This example matches the settings shown in the [NXLog configuration](#) above.

System » SNMP : Traps : Destination » New Trap Record...

Record Properties	
Version	v3
Destination	192.168.6.143
Port	162
Network	Default
Security Level	Auth & Privacy
Security Name	public
Engine ID	
Authentication Protocol	SHA
Authentication Password	*****
Privacy Protocol	AES
Privacy Password	*****

Cancel **Repeat** **Finished**

5. In case of a High Availability (HA) group, synchronize the configuration changes to the other units.
 - a. Click on the yellow **Changes Pending** in the top left corner.
 - b. Select the Active unit which should be marked as **(Self)**.
 - c. Make sure the **Sync Device to Group** option is chosen and click [**Sync**].

The screenshot shows the F5 BIG-IP Device Management Overview page. At the top, there's a header bar with the F5 logo and status indicators: Firewall: Consistent, ONLINE (ACTIVE), Changes Pending, and Managed by BIG-IQ. Below the header is a navigation bar with Main, Help, and About tabs. The main content area has tabs for Overview and Settings. Under Overview, there's a section for Device Groups with a table:

Name	Sync Status	Number of Devices	Device Group Type	Sync Type
Sync-failover	Changes Pending	2	Sync-Failover	Manual
SSLs	Green (Synced)	3	Sync-Only	Manual
datasync-global-dg	Green (Synced)	3	Sync-Only	Manual
device_trust_group	Green (Synced)	3	Sync-Only	Auto

Below the table is a Sync Summary section with three tabs: Status, Changes Pending, and Details. The Details tab shows a message: "Recommended action: Synchronize I-lb1 to group Sync-failover". To the right of this is a "Show Advanced View" link.

Under the Sync Summary is a Devices section with a table:

HA Status	Name	Sync Status	Configuration Time
Green (Active)	I-lb1 (Self)	Changes Pending	3/14/2017 21:12:13
Grey (Standby)	I-lb2	Green (Synced)	3/14/2017 20:31:43

At the bottom of the page are Sync Options (radio buttons for Sync Device to Group or Sync Group to Device) and an Overwrite Configuration checkbox. A Sync button is at the very bottom.

NOTE

Once the configuration has been synchronized to all members of the group, each member will be sending logs, inserting its own hostname and IP address. In the event of failover, logging will continue from both units regardless of which one is currently active.

49.4. BIG-IP High Speed Logging

F5 BIG-IP devices support High Speed Logging (HSL). This protocol will send as much data as the remote destination is able to accept. Combined with load balancing, BIG-IP makes it possible to have multiple NXLog servers load balanced with one of the available load balancing methods.

BIG-IP is able to send its own logs via HSL in addition to logs for traffic passing through the device. Because the load balancer is usually on the edge of the network and all web traffic passes through it, logging traffic on BIG-IP itself may be an easier and faster solution than processing web server logs on each server separately.

When configuring HSL on BIG-IP, the administrator will have to choose between sending logs via TCP or UDP. TCP can guarantee reliable delivery. However when load balancing traffic between multiple nodes, BIG-IP will reuse existing TCP connections to each node in order to reduce overhead related to creating new connections. This may result in less perfect load balancing between members.

NOTE

The steps below have been tested with BIG-IP v12.

In order to configure HSL on BIG-IP, a node for each NXLog server must be created and then added to a pool. Follow these steps.

1. Log in to BIG-IP via SSH.
2. Create a node for each NXLog agent.

```
# tmsh create ltm node NAME { address IP_ADDRESS session user-enabled }
```

3. Create a pool with all nodes.

```
# tmsh create ltm pool NAME { members add { NODE1:PORT { address \
IP_ADDRESS1 } } NODE2:PORT { address IP_ADDRESS2 } } monitor PROTOCOL }
```

Example 202. Creating a Pool

These commands create a pool named **nxlog** with one NXLog node.

```
# tmsh create ltm node nxlog1 { address 192.168.6.143 session user-enabled }
# tmsh create ltm pool nxlog { members add { nxlog1:1514 { address \
192.168.6.143 } } monitor tcp }
```

49.4.1. Forwarding BIG-IP Logs to an HSL Pool

To send logs generated on BIG-IP itself to the pool created above, follow these steps.

1. Log in to BIG-IP via SSH.
2. Create a remote logging destination. Replace **NAME** with a name for the destination, **POOL** with the name used above when creating the pool, **DISTRIBUTION** with one of the distribution options shown below, and **PROTOCOL** with **tcp** or **udp**. Distribution options include:

Adaptive

Sends traffic to one of the pool members until this member is either unable to process logs at the required rate or the connection is lost.

Balanced

Uses the load balancing method configured on the pool and selects a new member each time it sends data.

Replicated

Sends each log to all members of the pool.

```
# tmsh create sys log-config destination remote-high-speed-log NAME \
pool-name POOL distribution DISTRIBUTION protocol PROTOCOL
```

3. Create a log publisher. Replace **NAME** with a name for the publisher and **DESTINATION** with the destination name used in the previous step.

```
# tmsh create sys log-config publisher NAME destinations add {DESTINATION}
```

4. Create a log filter. Replace **NAME** with a name for the filter, **LEVEL** with the required logging level between Emergency and Debugging, **PUBLISHER** with the name used in the previous step, and **SOURCE** with a particular process running on BIG-IP (or **all**, which sends all logs).

```
# tmsh create sys log-config filter NAME level LEVEL \
publisher PUBLISHER source SOURCE
```

Example 203. Sending All Logs to the NXLog Pool

The following commands will send all logs to the NXLog pool via the TCP protocol.

```
# tmsh create sys log-config destination remote-high-speed-log nxlog-hsl \
  pool-name nxlog distribution adaptive protocol tcp
# tmsh create sys log-config publisher bigip-local-logs \
  destinations add {nxlog-hsl}
# tmsh create sys log-config filter bigip-all-local-logs level debug \
  publisher bigip-local-logs source all
```

49.4.2. Forwarding Traffic Logs to an HSL Pool

Configuring BIG-IP to log traffic that goes through the unit is done per virtual server and requires the following steps.

1. Configure NXLog (see the [examples](#) below), then restart NXLog.
2. Create a request logging profile. In most cases it is enough to log only requests, however if required the same profile can be configured to log responses and logging errors. Replace **NAME** with a name for the profile, **PROTOCOL** with **mds-tcp** or **mds-udp**, **POOL** with the pool name, and **TEMPLATE** with a list of HTTP request fields that will be logged (see the [LTM implementation guide](#)).

```
# tmsh create ltm profile request-log NAME { \
  request-log-protocol PROTOCOL request-log-pool POOL request-logging enabled \
  request-log-template "TEMPLATE" }
```

3. Assign the logging profile to a virtual server. Replace **NAME** with the virtual server name and **LOGGING_PROFILE** with the profile name used in the previous step. A logging profile can be assigned to multiple virtual servers.

```
# tmsh modify ltm virtual NAME {profiles add {LOGGING_PROFILE {}}}
```

Example 204. Logging Traffic to the NXLog Pool

The following commands configure traffic logging to the NXLog pool via TCP.

```
# tmsh create ltm profile request-log traffic-to-nxlog { \
  request-log-protocol mds-tcp request-log-pool nxlog request-logging enabled \
  request-log-template "client $CLIENT_IP:$CLIENT_PORT request $HTTP_REQUEST \
  server $SERVER_IP:$SERVER_PORT status $HTTP_STATUS" }
# tmsh modify ltm virtual q-web-farm-HTTPS {profiles add {traffic-to-nxlog {}}}
```

Example 205. Receiving Traffic Logs From BIG-IP

This example shows BIG-IP traffic logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_tcp>
10    Module im_tcp
11    Host   0.0.0.0
12    Port   1514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output out>
17    Module om_file
18    File   "/var/log/f5.log"
19    Exec   to_json();
20 </Output>
```

Below is an example of a request being logged in JSON format.

Output Sample

```
{
  "MessageSourceAddress": "192.168.6.91",
  "EventReceivedTime": "2017-05-10 19:16:43",
  "SourceModuleName": "in_syslog_tcp",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "USER",
  "SyslogSeverityValue": 5,
  "SyslogSeverity": "NOTICE",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2017-05-10 19:16:43",
  "Hostname": "192.168.6.91",
  "Message": "client 192.168.9.78:63717 request GET /cmmedia/img/icons/mime/mime-
unknown.png?v170509919 HTTP/1.1 server 192.168.6.101:80 status "
}
```

Example 206. Extracting Additional Fields

Further field extraction can be done with NXLog according to the sequence of fields specified in the template. For the template string shown above, the following configuration extracts the four fields.

nxlog.conf

```

1 <Input in_syslog_tcp>
2   Module  im_tcp
3   Host    0.0.0.0
4   Port    1514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /^client (.*) request (.*) server (.*) status (.*)$/
8     {
9       $HTTP_Client = $1;
10      $HTTP_Request = $2;
11      $HTTP_Server = $3;
12      $HTTP_Status = $4;
13    }
14  </Exec>
15 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.6.91",
  "EventReceivedTime": "2017-05-10 20:06:24",
  "SourceModuleName": "in_syslog_tcp",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "USER",
  "SyslogSeverityValue": 5,
  "SyslogSeverity": "NOTICE",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2017-05-10 20:06:24",
  "Hostname": "192.168.6.91",
  "Message": "client 192.168.9.78:65275 request GET /?disabledcookies=true HTTP/1.1 server 192.168.6.100:80 status ",
  "HTTP_Client": "192.168.9.78:65275",
  "HTTP_Request": "GET /?disabledcookies=true HTTP/1.1",
  "HTTP_Server": "192.168.6.100:80",
  "HTTP_Status": ""
}
```

49.4.3. Load Balancing Logs From External Sources via BIG-IP

Having a pool that is balancing traffic between multiple NXLog servers makes it possible to send logs from other servers and devices through BIG-IP. In order to accomplish this, create a virtual server that accepts Syslog traffic.

Example 207. Creating a Virtual Server Forwarding Logs to the NXLog Pool

This example creates a virtual server listening on TCP port 1514 that forwards logs to the **nxlog** pool.

```
# tmsh create ltm virtual nxlog-virtual-server { destination 192.168.6.93:1514 \
mask 255.255.255.255 pool nxlog profiles add { tcp{} } }
```

Once this has been set up, log producers can be configured to forward Syslog logs to 192.168.6.93.

Chapter 50. File Integrity Monitoring

File integrity monitoring (FIM) can be used to detect changes to files and directories. A file may be altered due to an update to a newer version, a security breach, or data corruption. File integrity monitoring helps an organization respond quickly and effectively to unexpected changes to files and is therefore a standard requirement for many regulatory compliance objectives.

- **PCI-DSS** - Payment Card Industry Data Security Standard (Requirement 11.5)
- **SOX** - Sarbanes-Oxley Act (Section 404)
- **NERC CIP** - NERC CIP Standard (CIP-010-2)
- **FISMA** - Federal Information Security Management Act (NIST SP800-53 Rev3)
- **HIPAA** - Health Insurance Portability and Accountability Act of 1996 (NIST Publication 800-66)
- **SANS** - SANS Critical Security Controls (Control 3)

NXLog can be configured to provide file (or Windows Registry) integrity monitoring. An event is generated for each detected modification. These events can then be used to [generate alerts](#) or be [forwarded](#) for storage and auditing.

There are various ways that monitoring can be implemented; these fall into two categories.

Checksum Monitoring

The [im_fim](#) and [im_regmon](#) modules (available with NXLog Enterprise Edition only) provide monitoring based on a cryptographic checksums. On the first run (when a file set or the registry is in a known secure state), a database of checksums is created. Subsequent scans are performed at regular intervals, and the checksums are compared. When a change is detected, an event is generated.

- The [im_fim](#) module is platform independent, available on all platforms supported by NXLog, and has no external dependencies. Similarly, the [im_regmon](#) module requires no configuration outside of NXLog to monitor the Windows Registry.
- If there are multiple changes between two scans, only the cumulative effect is logged. For example, if a file is deleted and a new file is created in its place before the next scan occurs, a single modification event will be generated.
- It is not possible to detect which user made a change because the filesystem/registry does not provide that information, and there may be multiple changes by different users between scans.

Real-Time Monitoring

Files (and the Windows Registry) can also be monitored in real-time with the help of kernel-level auditing, which does not require periodic scanning. This type of monitoring is platform specific.

- Kernel-level monitoring usually provides improved performance, especially for large file sets.
- All events are logged; the granularity of reporting is not limited by the scan interval (because there is no scanning involved).
- Reported events may be very detailed, and usually include information about which user made the change.

See the following sections for details about setting up file integrity monitoring on various platforms.

50.1. Monitoring on Linux

Checksum monitoring on Linux can be configured with the [im_fim](#) module.

NXLog must have permission to read the files that are to be monitored. Run NXLog as root, make sure the [nxlog](#) user or group has permission to read the files, or change the user/group under which NXLog runs. See the [User](#) and [Group](#) directives.

Example 208. Using im_fim on Linux

This configuration uses **im_fim** to monitor a common set of system directories containing configuration, executables, and libraries. The RIPEMD-160 hash function is selected and the scan interval is set to 3,600 seconds (1 hour).

nxlog.conf

```

1 <Input fim>
2   Module      im_fim
3   File        "/bin/*"
4   File        "/etc/*"
5   File        "/lib/*"
6   File        "/opt/nxlog/bin/*"
7   File        "/opt/nxlog/lib/*"
8   File        "/sbin/*"
9   File        "/usr/bin/*"
10  File        "/usr/sbin/*"
11  Exclude     "/etc/hosts.deny"
12  Exclude     "/etc/mtab"
13  Digest      rmd160
14  Recursive   TRUE
15  ScanInterval 3600
16 </Input>
```

NXLog will report scan activity in its internal log.

Internal Log

```

2017-06-14 11:44:53 INFO Module 'fim': FIM scan started
2017-06-14 11:45:00 INFO Module 'fim': FIM scan finished in 7.24 seconds. Scanned folders: 833
Scanned files: 5081 Read file bytes: 379166339
```

Output Sample

```
{
  "EventTime": "2017-06-14 11:57:33",
  "Hostname": "ubuntu-xenial",
  "EventType": "CHANGE",
  "Object": "FILE",
  "PrevFileName": "/etc/ld.so.cache",
  "PrevModificationTime": "2017-06-14 11:20:47",
  "FileName": "/etc/ld.so.cache",
  "ModificationTime": "2017-06-14 11:56:55",
  "PrevFileSize": 46298,
  "FileSize": 46971,
  "DigestName": "rmd160",
  "PrevDigest": "1dbe24a108c044153d8499f073274b7ad5507119",
  "Digest": "ec0bc108b7c9e5d9efde9cb1407b91e618d24c4",
  "EventReceivedTime": "2017-06-14 11:57:33",
  "SourceModuleName": "fim",
  "SourceModuleType": "im_fim"
}
```

See the [Linux Audit System](#) chapter for details about setting up kernel-level auditing. It is even possible to combine the **im_fim** and **im_linuxaudit** modules for redundant monitoring.

Monitoring on Windows

The **im_fim** module can be used on Windows for monitoring a file set.

Example 209. Using im_fim on Windows

This configuration monitors the program directories for changes. The scan interval is set to 1,800 seconds (30 minutes). The events generated by NXLog are similar to those shown in [Using im_fim on Linux](#).

nxlog.conf

```
1 <Input fim>
2   Module      im_fim
3   File        'C:\Program Files\*'
4   File        'C:\Program Files (x86)\*'
5   Exclude     'C:\Program Files\nxlog\data\*'
6   Recursive   TRUE
7   ScanInterval 1800
8 </Input>
```

The Windows Registry can be monitored with the **im_regmon** module.

Example 210. Using im_regmon on Windows

This configuration monitors all registry keys below the specified path. The keys are scanned every 60 seconds.

nxlog.conf

```
1 <Input registry>
2   Module      im_regmon
3   RegValue   'HKLM\Software\Policies\*'
4   Recursive   TRUE
5   ScanInterval 60
6 </Input>
```

NXLog will report scan activity in its internal log.

Internal Log

```
2018-01-31 04:01:12 INFO Module 'registry': Registry scan started
2018-01-31 04:01:12 INFO Module 'registry': Registry scan finished in 0.00 seconds. Scanned
registry keys: 77 Scanned registry values: 48 Read value bytes: 2396
```

Output Sample

```
{
  "EventTime": "2018-01-31 04:01:12",
  "Hostname": "WINAD",
  "EventType": "CHANGE",
  "RegistryValueName": "HKLM\\Software\\Policies\\Microsoft\\TPM\\OSManagedAuthLevel",
  "PrevValueSize": 4,
  "ValueSize": 4,
  "DigestName": "SHA1",
  "PrevDigest": "0aaaf76f425c6e0f43a36197de768e67d9e035abb",
  "Digest": "3c585604e87f855973731fea83e21fab9392d2fc",
  "Severity": "WARNING",
  "SeverityValue": 3,
  "EventReceivedTime": "2018-01-31 04:01:12",
  "SourceModuleName": "registry",
  "SourceModuleType": "im_regmon",
  "MessageSourceAddress": "10.8.0.121"
}
```

Real-time monitoring can be implemented with Windows security auditing (see [Security auditing](#) on Microsoft

Docs). Sysmon also implements file and registry monitoring with a system service and device driver; see the [Sysmon chapter](#). In both cases, the generated events can be collected from the EventLog with the [im_msvisatalog](#) module (see the [Windows EventLog chapter](#)).

Chapter 51. Graylog

Graylog is a popular open source log management tool with a GUI that uses Elasticsearch as a backend. It provides centralized log collection, analysis, searching, visualization, and alerting features. NXLog can be configured as a collector for Graylog, using one of the output writers provided by the `xm_gelf` module. In such a setup, NXLog acts as a forwarding agent on the client machine, sending messages to a Graylog node.

See the [Graylog documentation](#) for more information about configuring and using Graylog.

51.1. Configuring GELF UDP Collection

1. In the Graylog web interface, go to **System > Inputs**.
2. Select input type **GELF UDP** and click the **[Launch new input]** button.
3. Select the Graylog node for your input or make it global. Provide a name for the input in the **Title** textbox. Change the default port if needed. Use the **Bind address** option to limit the input to a specific network interface.

Launch new GELF UDP input

Global
Should this input start on all nodes

Node
2552c95f / graylog
On which node should this input start

Title
nxlog_udp
Select a name of your new input that describes it.

Bind address
0.0.0.0
Address to listen on. For example 0.0.0.0 or 127.0.0.1.

Port
12201
Port to listen on.

Receive Buffer Size (optional)
262144
The size in bytes of the recvBufferSize for network connections to this input.

Override source (optional)

The source is a hostname derived from the received packet by default. Set this if you want to override it with a custom string.

Decompressed size limit (optional)
8388608
The maximum number of bytes after decompression.

Buttons: Cancel Save

4. After saving, the input will appear shortly.

Local inputs 1 configured

nxlog_udp GELF UDP RUNNING

On node ★ 2552c95f / graylog

```
bind_address: 0.0.0.0
decompress_size_limit: 8388608
override_source: <empty>
port: 12201
recv_buffer_size: 262144
```

Throughput / Metrics
 1 minute average rate: 0 msg/s
 Network IO: ▾0B ▲0B (total: ▾24.1KB ▲0B)
 Empty messages discarded: 0

Example 211. Sending GELF via UDP

This configuration loads the *xm_gelf* extension module and uses the **GELF_UDP** output writer to send GELF messages via UDP.

nxlog.conf

```
1 <Extension _gelf>
2   Module      xm_gelf
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        "/var/log/messages"
8 </Input>
9
10 <Output out>
11   Module     om_udp
12   Host       127.0.0.1
13   Port       12201
14   OutputType GELF
15 </Output>
```

51.2. Configuring GELF TCP or TCP/TLS Collection

1. In the Graylog web interface, go to **System > Inputs**.
2. Select input type **GELF TCP** and click the **[Launch new input]** button.
3. Select the Graylog node for your input or make it global. Provide a name for the input in the **Title** textbox. Change the default port if needed. Use the **Bind address** option to limit the input to a specific network interface.

Launch new *GELF TCP* input

Global
Should this input start on all nodes

Node
2552c95f / graylog
On which node should this input start

Title
nxlog_tcp
Select a name of your new input that describes it.

Bind address
0.0.0.0
Address to listen on. For example 0.0.0.0 or 127.0.0.1.

Port
12201
Port to listen on.

Receive Buffer Size (optional)
1048576
The size in bytes of the recvBufferSize for network connections to this input.

4. To use TLS configuration, provide the **TLS cert file** and the **TLS private key file** (a password is required if the private key is encrypted). Check **Enable TLS**.

TLS cert file (optional)
/etc/ssl/certs/graylog-cert.pem
Path to the TLS certificate file

TLS private key file (optional)
/etc/ssl/private/graylog-private.pem
Path to the TLS private key file

Enable TLS (optional)
Accept TLS connections

TLS key password (optional)

The password for the encrypted key file.

5. After saving, the input will appear shortly.

Local inputs 2 configured

nxlog_tcp GELF TCP RUNNING

On node ★ 2552c95f / graylog

```
bind_address: 0.0.0.0
decompress_size_limit: 8388608
max_message_size: 2097152
override_source: <empty>
port: 12201
recv_buffer_size: 1048576
tcp_keepalive: false
tls_cert_file: <empty>
tls_client_auth: disabled
tls_client_auth_cert_file: <empty>
tls_enable: false
tls_key_file: <empty>
tls_key_password: *****
use_null_delimiter: true
```

Throughput / Metrics

1 minute average rate: 0 msg/s
 Network IO: ▷0B ▲0B {total: ▷0B ▲0B }
 Active connections: 0 (0 total)
 Empty messages discarded: 0

Example 212. Sending GELF via TCP

This configuration loads the *xm_gelf* extension module and uses the **GELF_TCP** output writer to send GELF messages via TCP.

nxlog.conf

```
1 <Extension _gelf>
2   Module      xm_gelf
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        "/var/log/messages"
8 </Input>
9
10 <Output out>
11   Module     om_tcp
12   Host       127.0.0.1
13   Port       12201
14   OutputType GELF_TCP
15 </Output>
```

Example 213. Sending GELF via TCP/TLS

This configuration loads the `xm_gelf` extension module and uses the `GELF_TCP` output writer with the `om_ssl` module to send GELF messages via TLS encrypted connection.

`nxlog.conf`

```

1 <Extension _gelf>
2   Module      xm_gelf
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        "/var/log/messages"
8 </Input>
9
10 <Output out>
11  Module      om_ssl
12  Host        127.0.0.1
13  Port        12201
14  CertFile    %CERTDIR%/graylog.crt
15  AllowUntrusted  TRUE
16  OutputType  GELF_TCP
17 </Output>
```

51.3. Collector Sidecar Configuration

Graylog Collector Sidecar is a lightweight configuration management system for different log collectors. It can be used to manage NXLog from the Graylog console. It supports GELF output via UDP, TCP, and TCP/TLS. The main advantage of using Sidecar is that everything is orchestrated from a single Graylog console.

1. Stop and disable the NXLog system service, as the NXLog process will be managed by Graylog. Install and configure the collector sidecar for the target system. The details can found in the [Graylog Collector Sidecar documentation](#).

`collector_sidecar.yml`

```

server_url: http://10.0.2.2:9000/api/
update_interval: 30
tls_skip_verify: true
send_status: true
list_log_files:
  - /var/log
node_id: graylog-collector-sidecar
collector_id: file:/etc/graylog/collector-sidecar/collector-id
log_path: /var/log/graylog/collector-sidecar
log_rotation_time: 86400
log_max_age: 604800
tags:
  - linux
  - apache
  - redis
backends:
  - name: nxlog
    enabled: true
    binary_path: /usr/bin/nxlog
    configuration_path: /etc/graylog/collector-sidecar/generated/nxlog.conf
```

2. Go to **System > Collectors**. After a successful sidecar installation, a new collector should appear.

Filter collectors:						Include inactive collectors
Name	Status	Operating System	Last Seen	Collector Id	Collector Version	
graylog-collector-sidecar	Running	Linux	a few seconds ago	217def9b-a735-4890-be33-b1e43fd4d53d	0.1.1	Show messages

3. Click the [Create configuration] button.

Filter Configurations

FilterReset
[Create configuration](#)

4. Apply a tag for the configuration.

Configuration tags

Manage tags for this configuration. Collectors using one of these tags will automatically apply this configuration.

Tags
x audit

[Update tags](#)

Select a tag or create new ones by typing their name.

5. Create a new output of the required type. See the [Configuring GELF UDP Collection](#) and [Configuring GELF TCP or TCP/TLS Collection](#) sections above.
6. Create an input for NXLog (for example, a file input).

Create Input audit_log

Name

Type a name for this input

Forward to (Required)

Choose the collector output that will forward messages from this input

Type

Choose the input type you want to configure

Path to Logfile

Location of the log file to use. Wildcards are supported in filenames, like '*' or '?'

Poll Interval

In seconds how frequently the collector will check for new files and new log entries

Save read position

Restore read position in case of a collector restart

Read since start

Instructs the collector to only read logs which arrived after nxlog was started

Recursive file lookup

Specifies whether input files should be searched recursively under subdirectories

Rename check

7. Go back to **System > Collectors** to verify the setup. If everything is fine the collector should be in the **Running** state.

Chapter 52. HP ProCurve

HP ProCurve switches are capable of sending their logs to a remote Syslog destination via UDP or TCP. When sending logs over the network it is recommended to use TCP as the more reliable protocol. With UDP there is a potential to lose entries, especially when there is a high volume of messages. It is also possible to send logs via TLS if additional security is required.

ProCurve Log Sample

```
I 03/17/17 18:06:15 ports: port B3 is Blocked by STP↵
I 03/17/17 18:06:15 ports: port B3 is now on-line↵
I 03/17/17 18:24:57 SNTP: updated time by -4 seconds↵
I 03/17/17 21:03:04 ports: port B3 is now off-line↵
I 03/18/17 02:00:53 SNTP: updated time by -4 seconds↵
I 03/18/17 09:36:49 SNTP: updated time by -4 seconds↵
I 03/18/17 17:00:45 SNTP: updated time by -4 seconds↵
I 03/18/17 23:34:25 mgr: SME TELNET from 192.168.9.78 - MANAGER Mode↵
```

The HP ProCurve web interface does not provide a way to configure an external Syslog server, so this must be done via the command line (see the following sections). For more details on configuring logging for HP ProCurve switches, refer to the HP ProCurve Management and Configuration Guide available from [HP Enterprise Support](#). The actual document depends on the model and firmware version in use.

WARNING

In case of multiple switches running in redundancy mode (such as VRRP or similar), each device must be configured separately as failover happens per VLAN and logging configuration is not synchronized.

NOTE

The steps below have been tested with HP 4000 series switches but should also work for 2000, 6000, and 8000 series devices.

1. Configure NXLog to receive log entries over the network and process them as Syslog (see [Accepting Syslog via UDP, TCP, or TLS](#) and the [TCP example](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from the switch.
3. Connect to the switch via SSH or Telnet.
4. Run the following commands to configure Syslog logging. Replace **LEVEL** with the logging level (**debug**, **major**, **error**, **warning**, or **info**). Replace **FACILITY** with the Syslog facility to be used for the logs. Replace **IP_ADDRESS** with the address of the NXLog agent; **PROTOCOL** with **udp**, **tcp**, or **tls**; and **PORT** with the required port. If **PORT** is omitted, the default will be used (514 for UDP, 1470 for TCP, or 6514 for TLS).

```
# configure
(config)# logging severity LEVEL
(config)# logging facility FACILITY
(config)# logging IP_ADDRESS PROTOCOL PORT
(config)# write memory
```

Example 214. Configuring Syslog Forwarding via TCP

The following commands configure the switch to send logs to 192.168.6.143 via the default TCP port. Only logs with **info** severity level and higher will be sent, and the **local5** Syslog facility will be used.

```
# configure
(config)# logging severity info
(config)# logging facility local5
(config)# logging 192.168.6.143 tcp
(config)# write memory
```

Example 215. Receiving ProCurve Logs via TCP

This example shows HP ProCurve logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_tcp>
10    Module im_tcp
11    Host   0.0.0.0
12    Port   1470
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/hp.log"
19    Exec   to_json();
20 </Output>
```

Events like those at the beginning of the chapter will result in the following output.

Output Sample

```
{
  "MessageSourceAddress": "192.168.10.3",
  "EventReceivedTime": "2017-03-18 19:32:02",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 21,
  "SyslogFacility": "LOCAL5",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "192.168.10.3",
  "EventTime": "2017-03-19 00:27:27",
  "SourceName": "mgr",
  "Message": " SME TELNET from 192.168.9.78 - MANAGER Mode"
}
```

Chapter 53. Linux Audit System

The Linux Audit system provides fine-grained logging of security related events. The system administrator configures rules to specify what events are logged. For example, rules may be configured for logging of:

- access of a specific file or directory,
- specific system calls,
- commands executed by a user,
- authentication events, or
- network access.

The Audit system architecture includes:

- a kernel component which generates events,
- the **auditd** daemon which collects events from the kernel component and writes them to a log file,
- the **audisp** dispatcher daemon which relays events to other applications for additional processing, and
- the **auditctl** control utility which provides configuration of the kernel component.

These tools are provided for reading the Audit log files:

- **aulast** prints out a listing of the last logged in users,
- **aulastlog** prints out the last login for all users of a machine,
- **aureport** produces summary reports of the Audit logs,
- **ausearch** searches Audit logs for events fitting given criteria, and
- **auvirt** prints a list of virtual machine sessions found in the Audit logs.

For more information about the Audit system, see the [System Auditing chapter](#) of the Red Hat Enterprise Linux Security Guide, the installed manual pages, and the [Linux Audit Documentation Project](#).

53.1. Audit Rules

The Audit system generates events according to Audit *rules*. These rules can be set dynamically with **auditctl** or stored persistently in **/etc/audit/rules.d**. Persistent rule files in **/etc/audit/rules.d** are automatically compiled to **/etc/audit/audit.rules** when auditd is initialized.

There are three types of rules: a **control** rule modifies Audit's behavior, a **file system** rule watches a file or directory, and a **system call** rule generates a log event for a particular system call. For more details about Audit rules, see the [Defining Audit Rules](#) page of the Red Hat Enterprise Linux Security Guide.

Common **control** rules include the following.

- **-b backlog**: Set the maximum number of audit buffers. This should be higher for busier systems or for heavy log volumes.
- **-D**: Delete all rules and watches. Normally used as the first rule.
- **-e [0..2]**: Temporarily disable auditing with **0**, enable it with **1**, or lock the configuration until the next reboot with **2** (used as the last rule).

Example 216. Control Rules

This is a set of basic rules, some form of which is likely to be found in any ruleset.

```
# Delete all rules (normally used first)
-D

# Increase buffers from default 64
-b 320

# Lock Audit rules until reboot (used last)
-e 2
```

To create a **file system** rule, use **-w path -p permissions -k key_name**.

- The **path** argument defines the file or directory to be watched.
- The **permissions** argument sets the kinds of accesses that are logged, and is a string containing one or more of **r** (read access), **w** (write access), **x** (execute access), and **a** (attribute change).
- The **key_name** argument is an optional tag for identifying the rule.

Example 217. A File System Rule

This rule watches **/etc/passwd** for modifications and tags these events with **passwd**.

```
-w /etc/passwd -p wa -k passwd
```

To create a **system call** rule, use **-a action,filter -S system_call -F field=value -k key_name**.

- The **action** argument can be either **always** (to generate a log entry) or **never** (to suppress a log entry). Generally, use **never** rules before **always** rules, because rules are matched from first to last.
- The **filter** argument is one of **task** (when a task is created), **exit** (when a system call exits), **user** (when a call originates from user space) or **exclude** (to filter events).
- The **system_call** argument specifies the system call by name, and can be repeated by using multiple **-S** flags.
- The **field=value** pair can be used to specify additional match options, and can also be used more than once.
- The **key_name** argument is an optional tag for identifying the rule.

Example 218. A System Call Rule

This rule generates a log entry when the system time is changed.

```
-a always,exit -F arch=b64 -S adjtimex -S settimofday -k system_time
```

The different types of rules are combined to form a ruleset.

Example 219. An Audit Rules File

This is a simple Audit ruleset based on the above examples.

/etc/audit/rules.d/audit.rules

```
# Delete all rules
-D

# Increase buffers from default 64
-b 320

# Watch /etc/passwd for modifications and tag with 'passwd'
-w /etc/passwd -p wa -k passwd

# Generate a log entry when the system time is changed
-a always,exit -F arch=b64 -S adjtimex -S settimofday -k system_time

# Lock Audit rules until reboot
-e 2
```

53.2. Using im_linuxaudit

NXLog Enterprise Edition includes an **im_linuxaudit** module for directly accessing the kernel component of the Audit System. With this module, NXLog can be configured to configure Audit rules and collect logs without requiring auditd or any other userspace software.

Example 220. Auditing With im_linuxaudit

This configuration uses a <Rules> block to specify a rule set.

nxlog.conf

```
1 <Input audit>
2   Module  im_linuxaudit
3   <Rules>
4     # Delete all rules (This rule has no affect; it is performed
5     # automatically by im_linuxaudit)
6     -D
7
8     # Increase buffers from default 64
9     -b 320
10
11    # Watch /etc/passwd for modifications and tag with 'passwd'
12    -w /etc/passwd -p wa -k passwd
13
14    # Generate a log entry when the system time is changed
15    -a always,exit -F arch=b64 -S adjtimex -S settimofday -k system_time
16
17    # Lock Audit rules until reboot
18    -e 2
19  </Rules>
20 </Input>
```

Example 221. Using a Separate Rules File With im_linuxaudit

This configuration is the same as the previous, but it uses a separate rules file. The referenced **audit.rules** file is identical to the one shown in the above [example](#), but it is stored in a different location (because auditd is not required).

nxlog.conf

```
1 <Input audit>
2   Module      im_linuxaudit
3   LoadRule    '/opt/nxlog/etc/audit.rules'
4 </Input>
```

53.3. Using auditd Userspace

There are also several ways to collect Audit logs via the regular Audit userspace tools, including [from auditd logs](#) and [by network via audisdp](#).

53.3.1. Setting up auditd

First, the Audit userspace components must be installed and configured.

1. Install the Audit package. Include the **audisdp-plugins** package if required for use with audisdp (see the [Collecting via Network With audisdp](#) section below).

- For RedHat/CentOS:

```
# yum install audit
```

- For Debian/Ubuntu:

```
# apt-get install auditd
```

2. Configure Auditd by editing the **/etc/audit/auditd.conf** configuration file, which contains parameters for auditd. See the [Configuring the Audit Service](#) page in the Red Hat Enterprise Linux Security Guide and the **auditd.conf(5)** man page.

3. After modifying the configuration or rules, enable or restart the auditd service to reload the configuration and update the rules (if they are not locked).

- For RedHat/CentOS:

```
# service auditd start
# systemctl enable auditd
```

- For Debian/Ubuntu:

```
# systemctl restart auditd
```

53.3.2. Reading auditd Logs

By default, auditd logs events to **/var/log/audit/audit.log** with root ownership. NXLog can be configured to read logs from that file.

1. NXLog cannot read logs owned as root when running as the **nxlog** user. Either omit the **User** option in **nxlog.conf** to run NXLog as root, or adjust the permissions as follows (see [Reading Rsyslog Log Files](#) for more information about **/var/log** permissions):

- a. use the `log_group` option in `/etc/audit/audit.conf` to set the group ownership for Audit log files,
 - b. change the current ownership of the log directory and files with `chgrp -R adm /var/log/audit`, and
 - c. add the `nxlog` user to the `adm` group with `usermod -a -G adm nxlog`.
2. Configure NXLog (see the example below) and restart.

Example 222. Reading From audit.log

In the Input block of this configuration, Audit logs are read from file, the key-value pairs are parsed with `xm_kvp`, and then some additional fields are added. In the Output block, the messages are converted to JSON format, BSD Syslog headers are added, and the logs are sent to another host via TCP.

nxlog.conf

```

1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Extension _syslog>
6   Module      xm_syslog
7 </Extension>
8
9 <Extension audit_parser>
10  Module      xm_kvp
11  KVPDelimiter ' '
12  KVDelimiter =
13  EscapeChar  '\'
14 </Extension>
15
16 <Input in>
17   Module      im_file
18   File        "/var/log/audit/audit.log"
19   <Exec>
20     audit_parser->parse_kvp();
21     $Hostname = hostname();
22     $FQDN = hostname_fqdn();
23     $Tag = "audit";
24     $SourceName = "selinux";
25   </Exec>
26 </Input>
27
28 <Output out>
29   Module      om_tcp
30   Host        192.168.1.1
31   Port        1514
32   Exec        to_json(); to_syslog_bsd();
33 </Output>
```

53.3.3. Collecting via Network With audispd

The Audit dispatcher (audispd) can be configured to forward log events to a remote server using the `audisp-remote` plugin included in the `audispd-plugins` package.

1. Configure the `audisp-remote` plugin. Use appropriate values for the `remote_server` and `format` directives.

/etc/audisp/audisp-remote.conf

```

remote_server = 127.0.0.1
port = 60
transport = tcp
queue_file = /var/spool/audit/remote.log
mode = immediate
queue_depth = 2048
format = ascii
network_retry_time = 1
max_tries_per_record = 3
max_time_per_record = 5
heartbeat_timeout = 0

network_failure_action = stop
disk_low_action = ignore
disk_full_action = ignore
disk_error_action = syslog
remote_ending_action = reconnect
generic_error_action = syslog
generic_warning_action = syslog
overflow_action = syslog

```

2. Activate the plugin by editing **/etc/audisp/plugins.d/au-remote.conf** and setting **active = yes**.
3. Optionally, audited may be configured to forward logs only (and not write to log files). Edit **/etc/audit/auditd.conf** and set **write_logs = no** (this option replaces **log_format = NOLOG**).
4. Configure NXLog (see the example below), then restart NXLog.
5. Restart the auditd service.

Example 223. Collecting via Network

With the following configuration, NXLog will accept Audit logs via TCP from audisdp on the local host, parse the key-value pairs with **xm_kvp**, and add some additional fields to the event record.

nxlog.conf

```

1 <Extension audit_parser>
2   Module      xm_kvp
3   KVPDelimiter ' '
4   KVDelimiter  =
5   EscapeChar   '\'
6 </Extension>
7
8 <Input in>
9   Module      im_tcp
10  Host        127.0.0.1
11  Port        60
12  <Exec>
13    audit_parser->parse_kvp();
14    $Hostname = hostname();
15    $FQDN = hostname_fqdn();
16    $Tag = "audit";
17    $SourceName = "auditd";
18  </Exec>
19 </Input>

```

Chapter 54. Linux System Logs

NXLog can be used to collect and process logs from a Linux system.

Linux distributions normally use a "Syslog" system logging agent to retrieve events from the kernel ([/proc/kmsg](#)) and accept log messages from user-space applications ([/dev/log](#)). Originally, this logger was **syslogd**; later **syslog-ng** added additional features, and finally **Rsyslog** is the logger in common use today. For more information about Syslog, see [Syslog](#).

Many modern Linux distributions also use the **Systemd** init system, which includes a **journal** component for handling log messages. All messages generated by Systemd-controlled processes are sent to the journal. The journal also handles messages written to [/dev/log](#). The journal stores logs in a binary format, either in memory or on disk; the logs can be accessed with the **journalctl** tool. Systemd can also be configured to forward logs via a socket to a local logger like Rsyslog or NXLog.

There are several ways that NXLog can be configured to collect Linux logs. See [Replacing Rsyslog](#) for details about replacing Rsyslog altogether, handling all logs with NXLog instead. See [Forwarding Messages via Socket](#) for a simple way to forward all logs to NXLog without disabling Rsyslog (this is the least intrusive option). Finally, it is also possible to read the log files written by Rsyslog; see [Reading Rsyslog Log Files](#).

54.1. Replacing Rsyslog

Follow these steps to disable Rsyslog and configure NXLog to collect logs in its place.

1. Configure NXLog to collect events from the kernel, the Systemd journal socket, and the [/dev/log](#) socket. See the [example](#) below.
2. Configure Systemd to forward log messages to a socket by enabling the **ForwardToSyslog** option.

/etc/systemd/journald.conf

```
[Journal]
ForwardToSyslog=yes
```

3. Stop and disable Rsyslog by running `systemctl stop rsyslog` and `systemctl disable rsyslog` as root.
4. Restart NXLog.
5. Reload the journald configuration by running `systemctl force-reload systemd-journald`.

Example 224. Replacing Rsyslog With NXLog

This example configures NXLog to read kernel events with the `im_kernel` module, read daemon messages from the Systemd journal socket with the `im_uds` module, and accept other user-space messages from the `/dev/log` socket with `im_uds`. In the `om_tcp` module instance, all of the logs are converted to JSON format, BSD Syslog headers are added, and the logs are forwarded to another host via TCP.

`nxlog.conf`

```
1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Extension _syslog>
6   Module      xm_syslog
7 </Extension>
8
9 <Input kernel>
10  Module     im_kernel
11  Exec       parse_syslog_bsd();
12 </Input>
13
14 <Input journal>
15  Module     im_uds
16  UDS       /run/systemd/journal/syslog
17  Exec       parse_syslog_bsd();
18 </Input>
19
20 <Input devlog>
21  Module     im_uds
22  UDS       /dev/log
23  FlowControl FALSE
24  Exec       $raw_event =~ s/\s+$//; parse_syslog_bsd();
25 </Input>
26
27 <Output out>
28  Module     om_tcp
29  Host      192.168.1.1
30  Port      1514
31  Exec       $Message = to_json(); to_syslog_bsd();
32 </Output>
33
34 <Route r>
35  Path      kernel, journal, devlog => out
36 </Route>
```

NOTE

Some local Syslog sources will add a trailing newline (`\n`) to each log message. The `$raw_event =~ s/\s+$//;` statement in the `devlog` input section above will automatically remove this and any other trailing whitespace before processing the message.

54.2. Forwarding Messages via Socket

By adding a short configuration file, Rsyslog can be configured to forward messages to NXLog via a Unix domain socket. This is the least intrusive of the options documented here.

1. Configure NXLog to accept log messages from Rsyslog via a socket. See the [example](#) below.

2. Configure Rsyslog to write to the socket by adding the following configuration file. See the [Rsyslog documentation](#) for more information about configuring what is forwarded to NXLog.

`/etc/rsyslog.d/nxlog.conf`

```
# Load omuxsock module
$ModLoad omuxsock

# Set socket path
$OMUXSockSocket /opt/nxlog/var/spool/nxlog/rsyslog_sock

# Configure template to preserve PRI part (must be on a single line)
$template SyslogWithPRI, "<%PRI%>%timegenerated% %HOSTNAME% %syslogtag%msg:::drop-last-1f%"

# Forward all log messages
.* :omuxsock:;SyslogWithPRI

# Only forward log messages of "notice" priority and higher
#.notice :omuxsock:;SyslogWithPRI
```

3. Restart NXLog and Rsyslog in that order to create and use the socket (NXLog must create the socket before Rsyslog will write to it). Run `sudo systemctl restart nxlog` and `sudo systemctl restart rsyslog`.

Example 225. Collecting Logs via Socket From Rsyslog

With this example configuration, NXLog will create the socket and accept log messages from Rsyslog through the socket. The messages will then be parsed as Syslog, converted to JSON format, prefixed with a BSD Syslog header, and forwarded to another host via TLS.

`nxlog.conf`

```
1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Extension _syslog>
6   Module      xm_syslog
7 </Extension>
8
9 <Input in>
10  Module     im_uds
11  UDS        /opt/nxsec/var/spool/nxlog/rsyslog_sock
12  Exec       parse_syslog();
13 </Input>
14
15 <Output out>
16  Module     om_ssl
17  Host       192.168.1.1
18  Port       6514
19  CAFile    %CERTDIR%/ca.pem
20  CertFile  %CERTDIR%/client-cert.pem
21  CertKeyFile %CERTDIR%/client-key.pem
22  Exec       $Message = to_json(); to_syslog_bsd();
23 </Output>
```

54.3. Reading Rsyslog Log Files

NXLog can be configured to read from log messages written by Rsyslog, `/var/log/messages` for example. This is a slightly more intrusive option than the steps given in [Forwarding Messages via Socket](#).

NOTE

NXLog will not have access to the facility and severity codes because Rsyslog, by default, follows the BSD Syslog convention of not writing the PRI code to the `/var/log/messages` file.

By default, NXLog runs as user `nxlog` and does not have permission to read files in `/var/log`. The simplest solution for this is to run NXLog as root by omitting the `User` option, but it is more secure to provide the necessary permissions explicitly.

1. Check the user or group ownership of the files in `/var/log` and configure if necessary. Some distributions use a group for the log files by default. On Debian/Ubuntu, for example, Rsyslog is configured to use the `adm` group. Otherwise, modify the Rsyslog configuration to use different ownership for log files as shown below.

`/etc/rsyslog.conf` or `/etc/rsyslog.d/nxlog.conf`

```
$FileOwner root
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022

# Default on Debian/Ubuntu
$FileGroup adm

# Or use the "nxlog" group directly
#$FileGroup nxlog
```

2. Run NXLog under a user or group that has permission to read the log files. Either use a user or group directly with the `User` or `Group` option in `nxlog.conf`, or add the `nxlog` user to a group that has permission. For example, on Debian/Ubuntu add the `nxlog` user to the `adm` group by running `usermod -a -G adm nxlog`.
3. If necessary, fix permissions for any files NXLog will be reading from that already exist (use the correct group for your system).

```
# chgrp adm /var/log/messages
# chmod g+r /var/log/messages
```

4. Configure NXLog to read from the required file(s) (see the [example](#) below). Then restart NXLog.
5. If the Rsyslog configuration has been modified, restart Rsyslog (`systemctl restart rsyslog`).

Example 226. Reading Rsyslog Log Files

With the following configuration, NXLog will read logs from `/var/log/messages`, parse the events as Syslog, convert them to JSON, and forward the plain JSON to another host via TCP.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Extension _syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input in>
10    Module im_file
11    File   '/var/log/messages'
12    Exec   parse_syslog();
13 </Input>
14
15 <Output out>
16    Module om_tcp
17    Host   192.168.1.1
18    Port   1514
19    Exec   $raw_event = to_json();
20 </Output>
```

Chapter 55. Log Event Extended Format (LEEF)

NXLog Enterprise Edition can be configured to collect or forward logs in the LEEF format.

The [LEEF](#) log format is used by IBM Security QRadar products and supports Syslog as a transport. It describes an event using key-value pairs, and provides a list of predefined event attributes. Additional attributes can be used for specific applications.

Basic LEEF Syntax

```
SYSLOG_HEADER|LEEF_HEADER|EVENT_ATTRIBUTES←
```

The LEEF_HEADER part contains the following pipe-delimited fields.

- LEEF version
- Vendor
- Product name
- Product version
- Event ID
- Optional delimiter character, as the character or its hexadecimal value prefixed by `0x` or `x` (LEEF version 2.0)

The EVENT_ATTRIBUTES part contains a list of key-value pairs separated by a tab or the delimiter specified in the LEEF header.

Full LEEF Syntax

```
Oct 11 11:27:23 myserver LEEF:Version|Vendor|Product|Version|EventID|Delimiter|src=192.168.1.1 →  
dst=10.0.0.1←
```

55.1. Collecting LEEF Logs

NXLog Enterprise Edition can parse LEEF logs with the [parse_leef\(\)](#) procedure provided by the *xm_leef* extension module.

Example 227. Accepting LEEF Logs via TCP

With the following configuration, NXLog will accept LEEF logs via TCP, convert them to JSON, and output the result to file.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Extension _leef>
6     Module xm_leef
7 </Extension>
8
9 <Input in>
10    Module im_tcp
11    Host   0.0.0.0
12    Port   1514
13    Exec   parse_leef();
14 </Input>
15
16 <Output out>
17    Module om_file
18    File   '/var/log/json'
19    Exec   to_json();
20 </Output>
```

Input Sample

```
Oct 11 11:27:23 myserver LEEF:2.0|Microsoft|MSEExchange|2013 SP1|15345|src=10.50.1.1 →
dst=2.10.20.20 ↳ spt=1200←
```

Output Sample

```
{
  "EventReceivedTime": "2016-10-11 11:27:24",
  "SourceModuleName": "in",
  "SourceModuleType": "im_file",
  "Hostname": "myserver",
  "LEEFVersion": "LEEF:2.0",
  "Vendor": "Microsoft",
  "SourceName": "MSEExchange",
  "Version": "2013 SP1",
  "EventID": "15345"
}
```

55.2. Generating LEEF Logs

NXLog Enterprise Edition can also generate LEEF logs, using the `to_leef()` procedure provided by the `xm_leef` extension module.

Example 228. Sending LEEF Logs via TCP

With this configuration, NXLog will parse the input JSON format from file and forward it as LEEF via TCP.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Extension _leef>
6     Module xm_leef
7 </Extension>
8
9 <Input in>
10    Module im_file
11    File   '/var/log/json'
12    Exec   parse_json();
13 </Input>
14
15 <Output out>
16    Module om_tcp
17    Host   10.12.0.1
18    Port   514
19    Exec   to_leef();
20 </Output>
```

Input Sample

```
{  
    "EventTime": "2016-09-13 11:23:11",  
    "Hostname": "myserver",  
    "Purpose": "test",  
    "Message": "This is a test log message."  
}
```

Output Sample

```
<13>Sep 13 11:23:11 myserver LEEF:1.0|NXLog|in|3.0.1775|unknown|EventReceivedTime=2016-09-13  
11:23:12 => SourceModuleName=in => SourceModuleType=im_file => devTime=2016-09-13 11:23:11 =>  
identHostName=myserver => Purpose=test => Message=This is a test log message. =>  
devTimeFormat=yyyy-MM-dd HH:mm:ss<
```

Chapter 56. Microsoft Exchange

Microsoft Exchange is a widely used enterprise level email server running on Microsoft Windows Server operating systems. The following sections describe various logs generated by Exchange and provide solutions for collecting logs from these sources with NXLog.

Exchange stores most of its operational logs in a comma-delimited format similar to W3C. These files can be read with [im_file](#) and the [xm_w3c](#) extension module. For NXLog Community Edition, the [xm_csv](#) extension module can be used instead, with the fields listed explicitly and the header lines skipped. In some of the log files, the W3C header is prepended by an additional CSV header line enumerating the same fields as the `#Fields` directive; NXLog must be configured to skip that line also. See the sections under [Transport Logs](#) for examples.

The information provided here is not intended to be comprehensive, but rather provides a general overview of NXLog integration with some of the major log mechanisms used by Exchange. Other logs generated by Exchange can be found in the [Logging](#) and other subdirectories of the installation directory.

NOTE

This Guide focuses on Exchange Server 2010 SP1 and later versions. Older versions are either not supported by Microsoft or are being decommissioned. Apart from passing their end of life date, these versions also lack the audit logging feature.

56.1. Transport Logs

Exchange Server writes various transport logs. Three of those logs are covered in the following sections. For more information about additional Exchange transport logs, see the [Transport logs in Exchange 2016](#) TechNet article.

56.1.1. Configuring Transport Logs

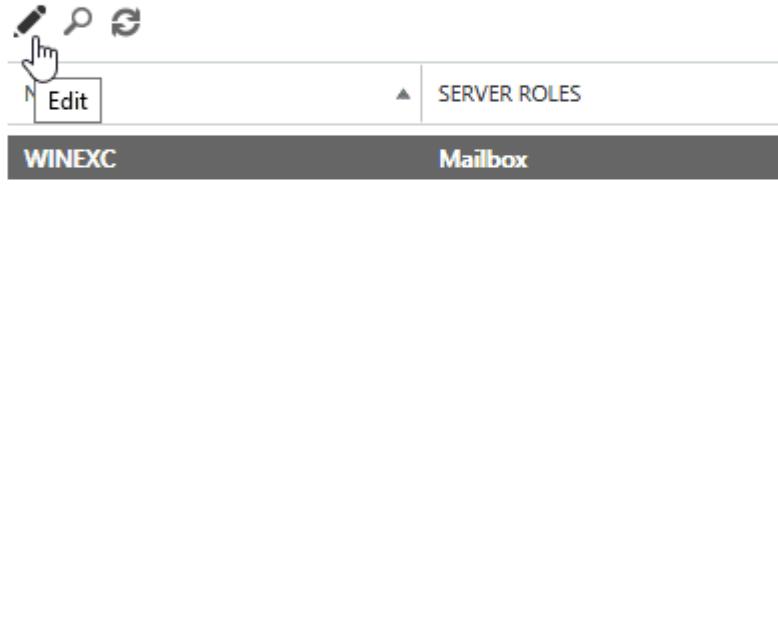
Message tracking, connectivity, and protocol logs are enabled by default and written to comma-delimited log files, in a format similar to W3C. The logs can be enabled or disabled, and the log file locations modified, through the Exchange Admin Center (EAC).

1. Log in to the Exchange Admin Center (at <https://server/ecp>).
2. Click **servers** in the list on the left.
3. Select the server and click the **Edit** icon.

Exchange admin center

- recipients
- permissions
- compliance management
- organization
- protection
- mail flow
- mobile
- public folders
- unified messaging
- servers**
- hybrid

servers databases database availability groups



4. Click **transport logs** in the list on the left.

general

databases and database availability groups

POP3

IMAP4

unified messaging

DNS lookups

transport limits

▶ **transport logs**

Outlook Anywhere

Message tracking log

Enable message tracking log

Message tracking log path:
C:\Program Files\Microsoft\Exchange Server\V15\Transport

Connectivity log

Enable connectivity log

Connectivity log path:
C:\Program Files\Microsoft\Exchange Server\V15\Transport

Protocol log

Send protocol log path:
C:\Program Files\Microsoft\Exchange Server\V15\TransportRole

Receive protocol log path:
C:\Program Files\Microsoft\Exchange Server\V15\TransportRole

5. Modify the logging configuration as required, then click **[Save]**.

56.1.2. Message Tracking Logs

Message tracking logs provide a detailed record of message activity as mail flows through the transport pipeline on an Exchange server.

Log Sample

```
#Software: Microsoft Exchange Server<
#Version: 15.01.1034.026<
#Log-type: Message Tracking Log<
#Date: 2017-09-15T20:01:45.863Z<
#Fields: date-time,client-ip,client-hostname,server-ip,server-hostname,source-context,connector-id,source,event-id,internal-message-id,message-id,network-message-id,recipient-address,recipient-status,total-bytes,recipient-count,related-recipient-address,reference,message-subject,sender-address,return-path,message-info,directionality,tenant-id,original-client-ip,original-server-ip,custom-data,transport-traffic-type,log-id,schema-version<
2017-09-15T20:01:45.863Z,,,WINEXC,No suitable shadow
servers,,SMTP,HAREDIRECTFAIL,34359738369,<49b4b9a2781a45cba555008075f7bffa@test.com>,8e1061b7-a376-497c-3172-
08d4fc7497bf,test1@test.com,,6533,1,,,test,Administrator@test.com,Administrator@test.com,,Originating,,,S:DeliveryPriority=Normal;S:AccountForest=test.com,Email,63dc9d79-5b4e-4f6c-1358-
08d4fc7497c3,15.01.1034.026<
```

NXLog can be configured to collect these logs with the [im_file](#) module, and to parse them with [xm_w3c](#).

Example 229. Collecting Message Tracking Logs With xm_w3c

This configuration collects message tracking logs from the defined **BASEDIR** and parses them using the [xm_w3c](#) module. The logs are then converted to JSON format and forwarded via TCP.

nxlog.conf

```
1 define BASEDIR C:\Program Files\Microsoft\Exchange Server\V15
2
3 <Extension _json>
4     Module      xm_json
5 </Extension>
6
7 <Extension w3c_parser>
8     Module      xm_w3c
9     Delimiter   ,
10 </Extension>
11
12 <Input messagetracking>
13     Module      im_file
14     File        '%BASEDIR%\TransportRoles\Logs\MessageTracking\MSGTRK*.LOG'
15     InputType   w3c_parser
16 </Input>
17
18 <Output tcp>
19     Module      om_tcp
20     Host        10.0.0.1
21     Port        1514
22     Exec        to_json();
23 </Output>
```

For NXLog Community Edition, the [xm_csv](#) module can be configured to parse these files.

Example 230. Using xm_csv for Message Tracking Logs

This configuration uses the [xm_csv](#) module to parse the message tracking logs.

nxlog.conf

```
1 define BASEDIR C:\Program Files\Microsoft\Exchange Server\V15
2
3 <Extension csv_parser>
4     Module      xm_csv
5     Fields      date-time, client-ip, client-hostname, server-ip, server-hostname, \
6                  source-context, connector-id, source, event-id, \
7                  internal-message-id, message-id, network-message-id, \
8                  recipient-address, recipient-status, total-bytes, recipient-count, \
9                  related-recipient-address, reference, message-subject, \
10                 sender-address, return-path, message-info, directionality, \
11                 tenant-id, original-client-ip, original-server-ip, custom-data, \
12                 transport-traffic-type, log-id, schema-version
13 </Extension>
14
15 <Input messagetracking>
16     Module      im_file
17     File        '%BASEDIR%\TransportRoles\Logs\MessageTracking\MSGTRK*.LOG'
18     <Exec>
19         if $raw_event =~ /(^(\xEF\xBB\xBF)?(date-time,|#))/ drop();
20         else
21             {
22                 csv_parser->parse_csv();
23                 $EventTime = parsedate(${date-time});
24             }
25     </Exec>
26 </Input>
```

56.1.3. Connectivity Logs

Connectivity logging records outbound message transmission activity by the transport services on the Exchange server.

Log Sample

```
#Software: Microsoft Exchange Server<br/>
#Version: 15.0.0.0<br/>
#Log-type: Transport Connectivity Log<br/>
#Date: 2017-09-15T03:09:34.541Z<br/>
#Fields: date-time,session,source,Destination,direction,description<br/>
2017-09-15T03:09:33.526Z,,Transport,,*,service started; #MaxConcurrentSubmissions=20;<br/>
MaxConcurrentDeliveries=20; MaxSmtpOutConnections=Unlimited<br/>
```

NXLog can be configured to collect these logs with the [im_file](#) module, and to parse them with [xm_w3c](#).

Example 231. Collecting Connectivity Logs With xm_w3c

This configuration collects connectivity logs from the defined **BASEDIR** and parses them using the [xm_w3c](#) module.

nxlog.conf

```
1 define BASEDIR C:\Program Files\Microsoft\Exchange Server\V15
2
3 <Extension w3c_parser>
4   Module      xm_w3c
5   Delimiter   ,
6 </Extension>
7
8 <Input connectivity>
9   Module      im_file
10  File        '%BASEDIR%\TransportRoles\Logs\Hub\Connectivity\CONNECTLOG*.LOG'
11  InputType   w3c_parser
12 </Input>
```

For NXLog Community Edition, the [xm_csv](#) module can be configured to parse these files.

Example 232. Using xm_csv for Connectivity Logs

This configuration uses the [xm_csv](#) module to parse the connectivity logs.

nxlog.conf

```
1 define BASEDIR C:\Program Files\Microsoft\Exchange Server\V15
2
3 <Extension csv_parser>
4   Module      xm_csv
5   Fields      date-time, session, source, Destination, direction, description
6 </Extension>
7
8 <Input connectivity>
9   Module      im_file
10  File        '%BASEDIR%\TransportRoles\Logs\Hub\Connectivity\CONNECTLOG*.LOG'
11  <Exec>
12    if $raw_event =~ /(^(\xEF\xBB\xBF)?(date-time,|#))/ drop();
13    else
14    {
15      csv_parser->parse_csv();
16      $EventTime = parsedate(${date-time});
17    }
18  </Exec>
19 </Input>
```

56.1.4. Protocol/SMTP Logs

Protocol logging records the SMTP conversations that occur on Send and Receive connectors during message delivery.

Log Sample

```
#Software: Microsoft Exchange Server
#Version: 15.0.0.0
#Log-type: SMTP Send Protocol Log
#Date: 2017-09-20T21:00:47.866Z
#Fields: date-time,connector-id,session-id,sequence-number,local-endpoint,remote-
endpoint,event,data,context
2017-09-20T21:00:47.167Z,internet,08D5006A392BE443,0,,64.8.70.48:25,*,,attempting to connect
```

NXLog can be configured to collect these logs with the [im_file](#) module, and to parse them with [xm_w3c](#).

Example 233. Collecting Protocol Logs With xm_w3c

This configuration collects protocol logs from the defined **BASEDIR** and parses them using the [xm_w3c](#) module.

nxlog.conf

```
1 define BASEDIR C:\Program Files\Microsoft\Exchange Server\V15
2
3 <Extension w3c_parser>
4   Module      xm_w3c
5   Delimiter ,
6 </Extension>
7
8 <Input smtp_receive>
9   Module      im_file
10  File        '%BASEDIR%\TransportRoles\Logs\Hub\ProtocolLog\SmtpReceive\RECV*.LOG'
11  InputType   w3c_parser
12 </Input>
13
14 <Input smtp_send>
15  Module      im_file
16  File        '%BASEDIR%\TransportRoles\Logs\Hub\ProtocolLog\SmtpSend\SEND*.LOG'
17  InputType   w3c_parser
18 </Input>
```

For NXLog Community Edition, the [xm_csv](#) module can be configured to parse these files.

Example 234. Using xm_csv for Protocol Logs

This configuration uses the [xm_csv](#) module to parse the protocol logs.

nxlog.conf

```
1 define BASEDIR C:\Program Files\Microsoft\Exchange Server\V15
2
3 <Extension csv_parser>
4     Module      xm_csv
5     Fields      date-time, connector-id, session-id, sequence-number, \
6                  local-endpoint, remote-endpoint, event, data, context
7 </Extension>
8
9 <Input smtp_receive>
10    Module   im_file
11    File    '%BASEDIR%\TransportRoles\Logs\Hub\ProtocolLog\SmtpReceive\RECV*.LOG'
12    <Exec>
13        if $raw_event =~ /(^xEF\xBB\xBF)?(date-time,|#)/ drop();
14        else
15        {
16            csv_parser->parse_csv();
17            $EventTime = parsedate(${date-time});
18        }
19    </Exec>
20 </Input>
21
22 <Input smtp_send>
23    Module   im_file
24    File    '%BASEDIR%\TransportRoles\Logs\Hub\ProtocolLog\SmtpSend\SEND*.LOG'
25    <Exec>
26        if $raw_event =~ /(^xEF\xBB\xBF)?(date-time,|#)/ drop();
27        else
28        {
29            csv_parser->parse_csv();
30            $EventTime = parsedate(${date-time});
31        }
32    </Exec>
33 </Input>
```

56.2. EventLog

Exchange Server also logs events to Windows EventLog. Events are logged to the Application and Systems channels, as well as multiple Exchange-specific crimson channels (see your server's Event Viewer). For more information about events generated by Exchange, see the following TechNet articles.

- [Error and Event Reference for Client Access Servers](#)
- [Error and Event Reference for Mailbox Servers](#)
- [Error and Event Reference for Transport Servers](#)
- [Error and Event Reference for Unified Messaging Servers](#)
- [Manage Diagnostic Logging Levels](#)
- [Managed Availability](#)
- [Messaging records management errors and events](#)
- [Monitoring database availability groups](#)

See also [Windows EventLog](#) for more information about using NXLog to collect logs from Windows EventLog.

Example 235. Collecting Exchange Events From the EventLog

With this configuration, NXLog will use the `im_msvistalog` module to subscribe to the Application and System channels (Critical, Error, and Warning event levels only) and the MSExchange Management crimson channel (all event levels). Note that the Application and System channels will include other non-Exchange events.

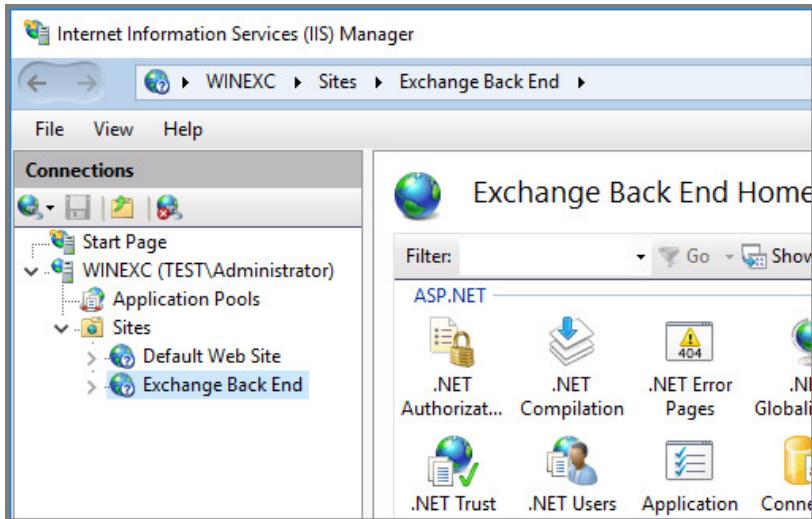
`nxlog.conf`

```

1 <Input eventlog>
2   Module im_msvistalog
3   <QueryXML>
4     <QueryList>
5       <Query Id="0" Path="Application">
6         <Select Path="Application">
7           *[System[(Level=1 or Level=2 or Level=3)]]</Select>
8         <Select Path="System">
9           *[System[(Level=1 or Level=2 or Level=3)]]</Select>
10        <Select Path="MSExchange Management">*</Select>
11      </Query>
12    </QueryList>
13  </QueryXML>
14 </Input>
```

56.3. IIS Logs

Exchange is closely integrated with the Internet Information Server (IIS), which itself logs Outlook Web Access (OWA) and Exchange Admin Center (EAC) events.



See the [Microsoft IIS](#) chapter for more information about collecting events from IIS with NXLog.

56.4. Audit Logs (nxlog-xchg)

Exchange also provides two types of audit logs: administrator audit logging and mailbox audit logging. For more information, see [Administrator audit logging in Exchange 2016](#) and [Mailbox audit logging in Exchange 2016](#) on TechNet.

The `nxlog-xchg` utility can be used to retrieve these logs. This utility is capable of logging actions taken by users or administrators who make changes in the organization, mailbox actions, and logins including by users other than the mailbox owner. It periodically queries the Exchange server for such logs and writes the result to standard output in JSON format for further processing by NXLog. The `nxlog-xchg` utility is executed by NXLog via the `im_exec` module, and may be configured on either the Exchange server itself or another system.

Server-side requirements include:

- Microsoft Exchange Server 2010 SP1+, 2013 or 2016;
- Windows Remoting (WinRM) with HTTPS listener (by default, WinRM listens for HTTP requests only);
- an Active Directory user that can log into the Windows server running Exchange; and
- an Active Directory user with at least the Organization Management and Records ManagementExchange administrator roles (may be the same user as the previous).

NOTE

The required steps may vary from those provided below based on the organization and domain topology and configuration.

56.4.1. Add User(s)

Create the Active Directory user(s) specified above, so nxlog-xchg can log in to the Exchange server and retrieve the audit events. For more information about assigning permissions in Exchange Server, see the [Permissions](#) article on TechNet.

1. Log in to the Exchange Admin Center (at <https://server/ecp>).
2. Click **recipients** in the list on the left, click **mailboxes**, click the **New** icon, and then click **User mailbox**.

The screenshot shows the Exchange Admin Center interface. On the left, there is a sidebar with links: recipients (which is selected and highlighted in blue), permissions, compliance management, organization, protection, mail flow, and mobile. On the right, there is a main content area with tabs: mailboxes (selected and highlighted in blue), groups, resources, and contacts. Below the tabs is a toolbar with icons for New (+), Edit (pencil), Delete (trash), Search (magnifying glass), Filter (refresh), and More (...). A modal dialog box is open over the list of mailboxes. The dialog has two tabs: 'User mailbox' (selected) and 'Linked mailbox'. It contains a 'MAILBOX TYPE' dropdown set to 'User'. Below the tabs, there is a table with two rows. The first row contains 'test1 last1' and 'User'. The second row contains 'test2 test2' and 'User'. At the bottom of the dialog is a 'Save' button.

MAILBOX TYPE
User

3. Fill in the form to create a new user, then click **[Save]**. The credentials will be used by nxlog-xchg to connect to the Exchange server.

new user mailbox

Alias:
nxlog

Existing user
 New user

First name:

Initials:

Last name:

*Display name:
nxlog

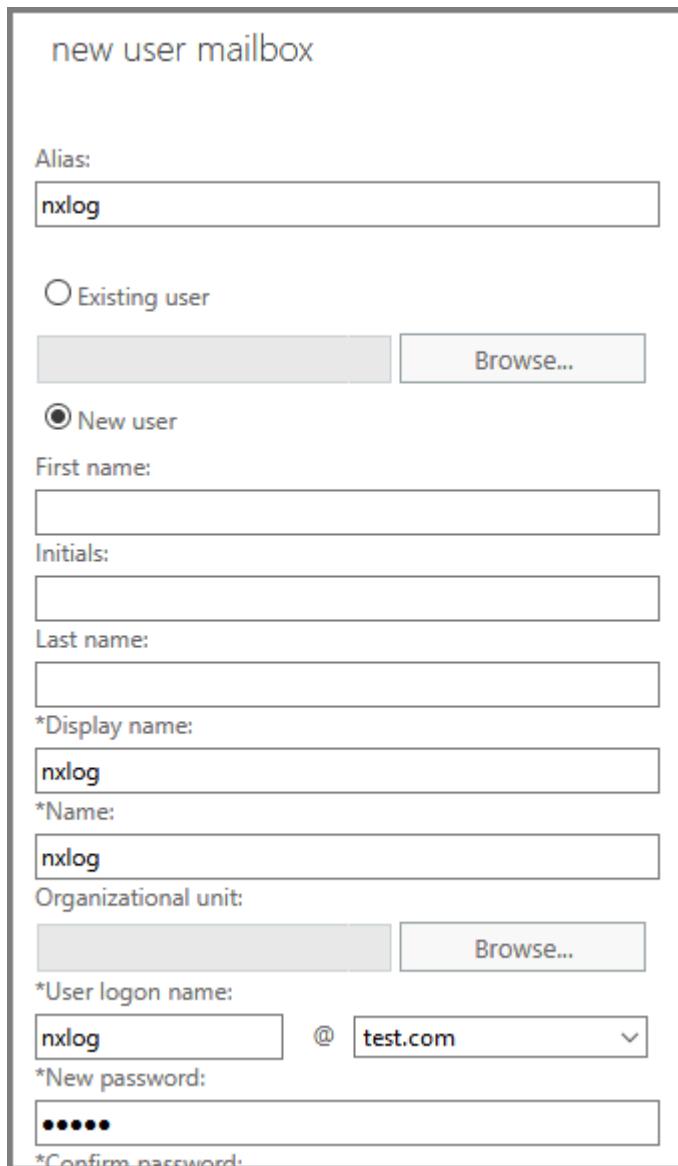
*Name:
nxlog

Organizational unit:

*User logon name:
nxlog @ test.com

*New password:

*Confirm password:



4. Click **recipients** in the list on the left, click **admin roles**, select the **Organization Management** role, and click the **Edit** icon.

The screenshot shows the Exchange admin center interface. On the left, there's a sidebar with categories like recipients, permissions (which is selected), compliance management, organization, protection, mail flow, mobile, public folders, and unified messaging. The main area is titled 'admin roles' and shows a list of roles: Compliance Management, Delegated Setup, Discovery Management, Help Desk, Hygiene Management, Organization Management (which is highlighted with a dark gray background), Public Folder Management, Recipient Management, Records Management, and Security Administrator. Above the list is a toolbar with icons for add, edit, delete, and search.

5. Add the newly-created user to the list of **Members**, then click **[Save]**.
6. Edit the **Records Management** role and add the new user there also.
7. If necessary, give the new user permission to log in to the Exchange server. Or, create another user for this purpose.

56.4.2. Configure WinRM

On the Exchange server, Windows Remoting (WinRM) must be configured to accept HTTPS requests from nxlog-xchg on TCP port 5986.

1. Open PowerShell and run `winrm quickconfig -transport:https`. If you receive an error message about the machine not having an appropriate certificate, you will need to either issue a certificate or create a self-signed certificate; continue with the following steps. Otherwise, skip to step 4.
2. Create a certificate. Either use your certificate authority or create a self-signed certificate.
 - ° To create a self-signed certificate, run `$cert=New-SelfSignedCertificate -CertStoreLocation Cert:\LocalMachine\My -DnsName "<SERVER_HOSTNAME>"`.
 - ° Or, to list available certificates, run `Get-ChildItem -Path Cert:\LocalMachine\My`. Then select the appropriate certificate by running `$cert=Get-ChildItem -Path cert:\LocalMachine\My\<THUMBPRINT>`
3. Create the HTTPS listener by running `New-Item -Path WSMan:\localhost\Listener -Transport HTTPS -Address * -CertificateThumbPrint $cert.ThumbPrint -Force`
4. Enable PowerShell remoting by running `Enable-PSRemoting -SkipNetworkProfileCheck -Force`.
5. You may wish to remove the WinRM HTTP listener by running `Get-ChildItem WSMan:\localhost\Listener | Where -Property Keys -eq "Transport=HTTP" | Remove-Item -Recurse`.

56.4.3. Configure Audit Logging

The administrator audit logging and mailbox audit logging features must be enabled on the Exchange server. The commands below should be executed in the Exchange Management Shell.

- Administrator audit logging is enabled by default. Verify by running `Get-AdminAuditLogConfig | FL AdminAuditLogEnabled`. See [Manage administrator audit logging](#) for more details.
- To enable mailbox audit logging for a single user, run `Set-Mailbox -Identity <MailboxIdParameter> -AuditEnabled $true`. For more information, including more logging options, see [Enable or disable mailbox audit logging for a mailbox](#).
- To enable audit logging for all user mailboxes in the organization, run `Get-Mailbox -ResultSize Unlimited -Filter {RecipientTypeDetails -eq "UserMailbox"} | Set-Mailbox -AuditEnabled $true`.

56.4.4. Client Setup

The nxlog-xchg utility must be configured, then NXLog must be configured to run the nxlog-xchg executable.

56.4.4.1. nxlog-xchg Configuration

The nxlog-xchg utility can be configured either by arguments on the command line or by a configuration file. The two methods cannot be mixed.

The command line arguments use the same names as in the configuration file. Three arguments are offered by nxlog-xchg in addition to those in the configuration file.

- **--debug**: set debug verbosity, 0-3 (0 = none/default, 3 = verbose)
- **-c, --config**: set the configuration file path
- **--version**: show the version of the nxlog-xchg utility

Example 236. Using Command Line Arguments

This example lists shows the use of command line arguments with sample values.

```
nxlog-xchg.exe --Url https://exchange01.corp.local:5986 --User winrmuser  
--Password winrmuser_password --HostFQDN exchange01.corp.local  
--ExchangeUser exuser@local --ExchangePassword exuser_password
```

Example 237. Using a Configuration File

The command below uses the **-c** or **--config** parameter to specify a configuration file.

```
nxlog-xchg.exe -c "C:\Program Files (x86)\nxlog-xchg\nxlog-xchg.cfg"
```

The following arguments can be set on the command line or in the configuration file.

[NXLog] section:

SavePos

This optional boolean directive specifies whether the last record number should be saved when nxlog-xchg exits. The default is TRUE.

PollInterval

This optional directive specifies the time (in seconds) between polls. Valid values are 3-3600; the default is 30 seconds.

[WinRM] section:

Url

This specifies the URL of the WinRM listener (for example, <https://exchangeserver.mydomain.com:5986/wsman>).

User

This specifies the user that has permission to log on to the Windows running Exchange Server.

Password

This should contain the password of the user defined above (in WinRM/User).

CheckCertificate

This optional directive specifies whether the certificate should be validated. The default is TRUE (the certificate is validated).

[Exchange] section:

HostFQDN

This directive specifies the fully qualified name of the Exchange server (for example, <name.domain.tld>).

ExchangeUser

This specifies the user that has permission to query the Exchange Server.

ExchangePassword

This should contain the password of the user defined above (in Exchange/ExchangeUser).

[Options] section:

QueryAdminLog

This optional boolean directive specifies whether the administrator audit log should be queried. The default is TRUE (the administrator audit log is queried).

QueryMailboxLog

This optional boolean directive specifies whether the mailbox audit log should be queried. The default is TRUE (the mailbox audit log is queried).

ResultSize

This optional directive specifies the maximum number of log entries to retrieve. The default is 5000 entries.

Example 238. An nxlog-xchg Configuration File

This configuration shows the default values for optional arguments and sample values for required arguments.

nxlog-xchg.cfg

```
[NXLog]
SavePos=TRUE
PollInterval=30

[WinRM]
Url=https://exchange01.corp.local:5986/wsman
User=winrmuser@local
Password=winrmuser_password
CheckCertificate=TRUE

[Exchange]
HostFQDN=exchange01.corp.local
ExchangeUser=ex_user@local
ExchangePassword=exuser_password

[Options]
QueryAdminLog=TRUE
QueryMailboxLog=TRUE
ResultSize=5000
```

56.4.4.2. NXLog Configuration

To including nxlog-xchg in a working NXLog installation, simply configure an Input block with the **im_exec** module and specify nxlog-xchg as the external program.

Example 239. Sending Exchange Audit Logs to a File

This Input module instance uses the nxlog-xchg utility and a corresponding configuration file to collect audit logs from an Exchange server.

nxlog.conf

```
1 <Input in>
2   Module im_exec
3   Command "C:\Program Files (x86)\nxlog-xchg\nxlog-xchg.exe"
4   Arg    -c
5   Arg    C:\Program Files (x86)\nxlog-xchg\nxlog-xchg.cfg
6 </Input>
```

Chapter 57. Microsoft IIS

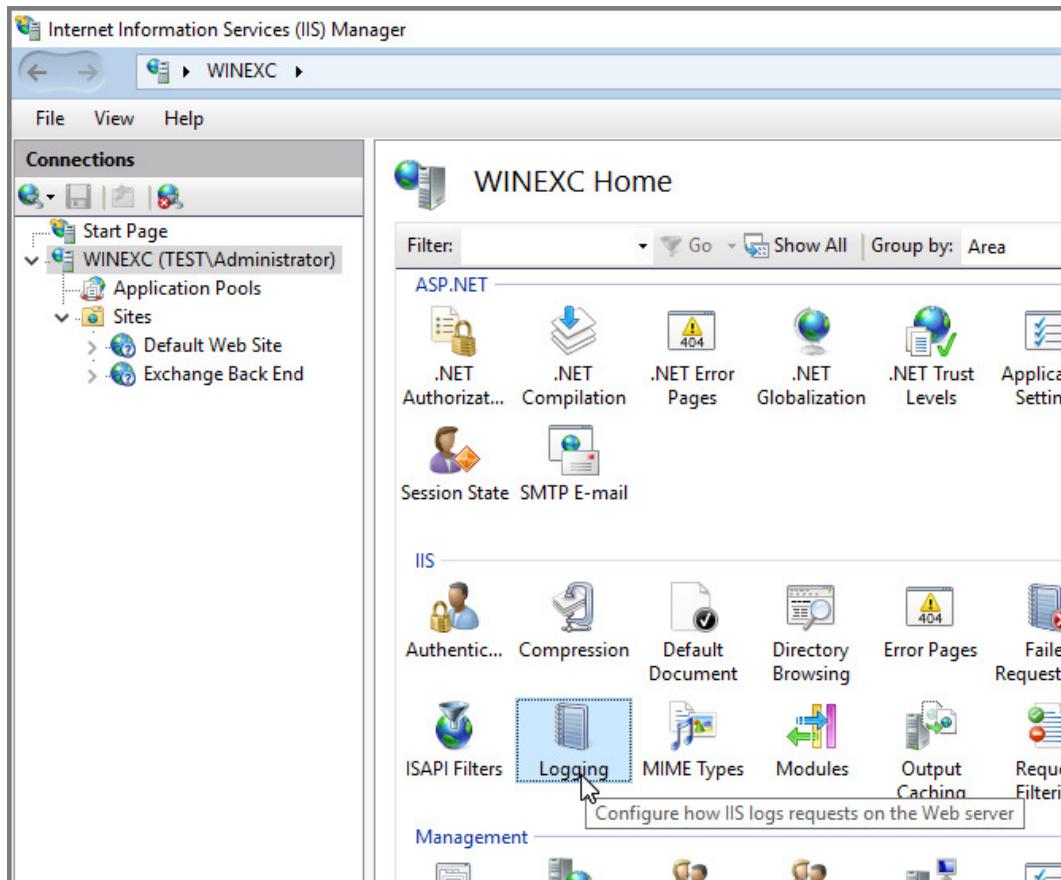
Microsoft Internet Information Server supports several logging formats. This chapter provides information about configuring IIS logging and NXLog collection. The recommended W3C format is documented below as well as other supported IIS formats.

This chapter also includes sections about collecting logs from the [SMTP Server](#) and about [Automatic Retrieval of IIS Site Log Locations](#).

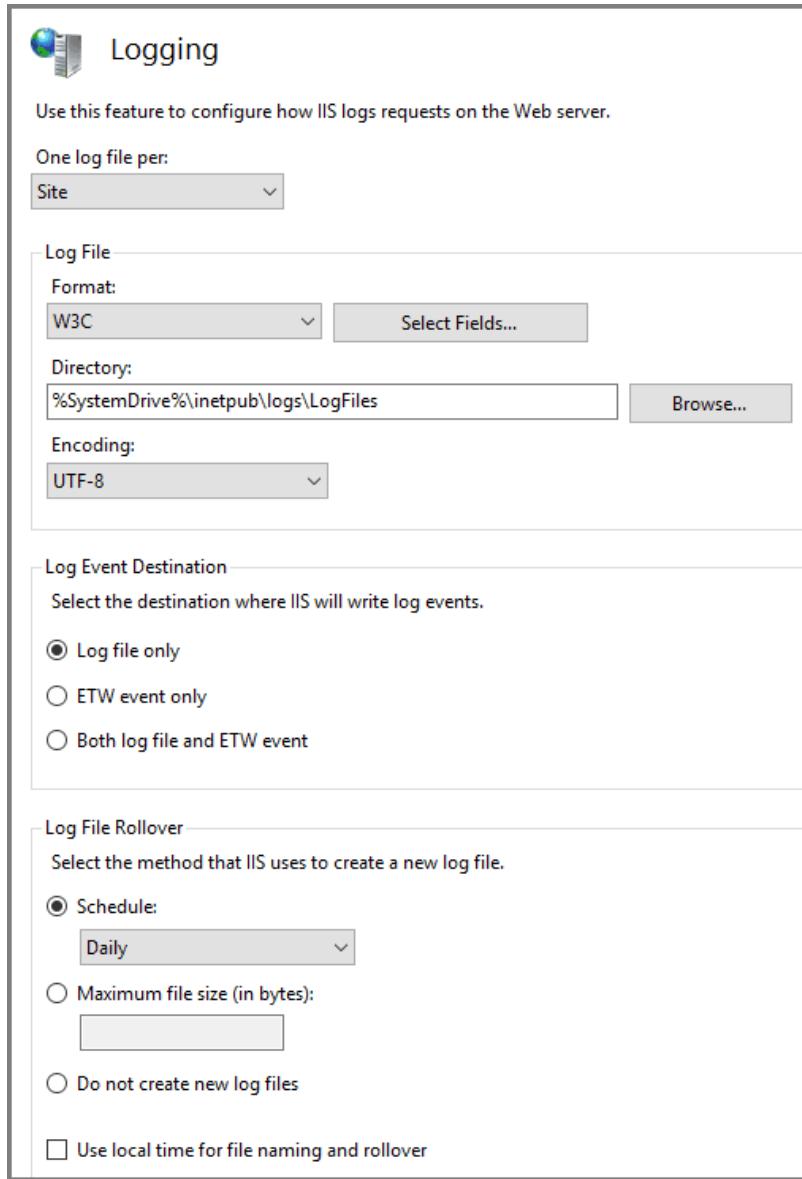
57.1. Configuring Logging

IIS logging can be configured at the site level or server level as follows. For more detailed information, see [Configure Logging in IIS](#) on Microsoft Docs.

1. Open **IIS Manager**, which can be accessed from the **Tools** menu in the **Server Manager** or from **Administrative Tools**.
2. In the **Connections** pane on the left, select the server or site for which to configure logging. Select a server to configure logging server-wide, or a site to configure logging for that specific site.
3. Double-click the **Logging** icon in the center pane.



4. Modify the logging configuration as required. The W3C format is recommended.



The resulting logs can be collected by NXLog as shown in the following sections.

57.2. W3C Extended Log File Format

IIS can write logs in the W3C format, and the logged fields can be configured via the [**Select Fields...**] button (see the [Configuring Logging](#) section). W3C is the recommended format for use with NXLog.

Log Sample

```
#Software: Microsoft Internet Information Services 10.0<
#Version: 1.0<
#Date: 2017-10-02 17:11:27<
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-ip cs(User-Agent)
cs(Referer) sc-status sc-substatus sc-win32-status time-taken<
2017-10-02 17:11:27 fe80::b5d8:132c:cec9:daef%6 RPC_IN_DATA /rpc/rpcproxy.dll 1d4026cb-6730-43bf-
91eb-df80f41c050f@test.com:6001&CorrelationID=<empty>;&RequestId=11d6a78a-7c34-4f43-9400-
ad23b114aa62&cafeReqId=11d6a78a-7c34-4f43-9400-ad23b114aa62; 80 TEST\HealthMailbox418406e
fe80::b5d8:132c:cec9:daef%6 MSRPC - 500 0 0 7990<
2017-10-02 17:12:57 fe80::a425:345a:7143:3b15%2 POST /powershell
clientApplication=ActiveMonitor;PSVersion=5.1.14393.1715 80 - fe80::a425:345a:7143:3b15%2
Microsoft+WinRM+Client - 500 0 0 11279<
```

Note that field names with special characters must be referenced with curly braces (for example, `${s-ip}` and

`${cs(User-Agent)}).`

See also the [W3C Extended Log File Format](#) section and the [W3C Extended Log File Format \(IIS 6.0\)](#) and [W3C Extended Log File Examples \(IIS 6.0\)](#) articles on Microsoft TechNet.

Example 240. Collecting W3C Format Logs With xm_w3c

This configuration reads from file with `im_file` and parses with `xm_w3c`.

nxlog.conf

```
1 <Extension w3c_parser>
2     Module      xm_w3c
3 </Extension>
4
5 <Input iis_w3c>
6     Module      im_file
7     File        'C:\inetpub\logs\LogFiles\W3SVC*\u_ex*.log'
8     InputType   w3c_parser
9 </Input>
```

For NXLog Community Edition, the `xm_csv` module can be used instead for parsing the records.

Example 241. Collecting W3C Format Logs With xm_csv

This configuration parses the logs with the `xm_csv` module. The header lines are discarded and the `$date` and `$time` fields are parsed in order to set an `$EventTime` field.

WARNING

The field list must be set according to the configured IIS fields. The fields shown here correspond with the default field selection in IIS versions 8.5 and 10.

nxlog.conf

```

1 <Extension w3c_parser>
2   Module      xm_csv
3   Fields      date, time, s-ip, cs-method, cs-uri-stem, cs-uri-query, \
4               s-port, cs-username, c-ip, cs(User-Agent), cs(Referer), \
5               sc-status, sc-substatus, sc-win32-status, time-taken
6   FieldTypes  string, string, string, string, string, string, integer, \
7               string, string, string, string, integer, integer, integer, \
8               integer
9   Delimiter   '
10  EscapeChar  '''
11  QuoteChar   '''
12  EscapeControl FALSE
13  UndefValue  -
14 </Extension>
15
16 <Input iis_w3c>
17   Module      im_file
18   File        'C:\inetpub\logs\LogFiles\W3SVC*\u_ex*.log'
19   <Exec>
20     if $raw_event =~ /^#/ drop();
21     else
22     {
23       w3c_parser->parse_csv();
24       $EventTime = parsedate($date + "T" + $time + ".000Z");
25     }
26   </Exec>
27 </Input>
```

57.3. IIS Log File Format

The IIS format is line-based, with comma-separated fields and no header. See [IIS Log File Format \(IIS 6.0\)](#) on TechNet for more information.

Log Sample

```

::1, HealthMailbox418406e8ac5b4b61a6b731ac4c660553@test.com, 9/28/2017, 14:49:00, W3SVC1, WINEXC,
::1, 7452, 592, 2538, 302, 0, POST, /OWA/auth.owa, &CorrelationID=<empty>;&cafeReqId=728beb5e-98de-
4680-acb2-45968bef533c;&encoding=,
127.0.0.1, -, 9/28/2017, 14:49:01, W3SVC1, WINEXC, 127.0.0.1, 6798, 2502, 682, 302, 0, GET, /ecp/,
&CorrelationID=<empty>;&cafeReqId=0ed28871-4083-492f-99c2-
2fbdb06a9466;&LogoffReason=NoCookiesGetOrE14AuthPost,
```

Example 242. Collecting Logs From the IIS Format

This configuration reads from file with `im_file` and parses the fields with `xm_csv`. The `$Date` and `$Time` fields are parsed in order to set an `$EventTime` field.

`nxlog.conf`

```
1 <Extension iis_parser>
2   Module      xm_csv
3   Fields      ClientIPAddress, UserName, Date, Time, ServiceAndInstance, \
4               ServerName, ServerIPAddress, TimeTaken, ClientBytesSent, \
5               ServerBytesSent, ServerStatusCode, WindowsStatusCode, RequestType, \
6               TargetOfOperation, Parameters
7   FieldTypes  string, string, string, string, string, string, string, integer, \
8               integer, integer, integer, string, string, string
9   UndefValue  -
10 </Extension>
11
12 <Input iis>
13   Module      im_file
14   File        'C:\inetpub\logs\LogFiles\W3SVC*\u_in*.log'
15   <Exec>
16     iis_parser->parse_csv();
17     $EventTime = strftime($Date + " " + $Time, "%m/%d/%Y %H:%M:%S");
18   </Exec>
19 </Input>
```

57.4. NCSA Common Log File Format

The NCSA format is a line-based plain text format that separates fields with spaces and uses hyphens (-) as placeholders for empty fields. See the [Common & Combined Log Formats](#) section for more information about this format. See [NCSA Common Log File Format \(IIS 6.0\)](#) on Microsoft TechNet for more information about this format as used by IIS.

Log Sample

```
fe80::a425:345a:7143:3b15%2 - - [02/Oct/2017:13:16:18 -0700] "POST
/mapi/emsmdb/?useMailboxOfAuthenticatedUser=true HTTP/1.1" 401 7226
fe80::a425:345a:7143:3b15%2 - TEST\HealthMailboxc0bafdf1 [02/Oct/2017:13:16:20 -0700] "POST
/mapi/emsmdb/?useMailboxOfAuthenticatedUser=true HTTP/1.1" 200 1482
```

Example 243. Collecting NCSA Format Logs

This configuration reads from file with the `im_file` module and uses a regular expression to parse each record.

nxlog.conf

```

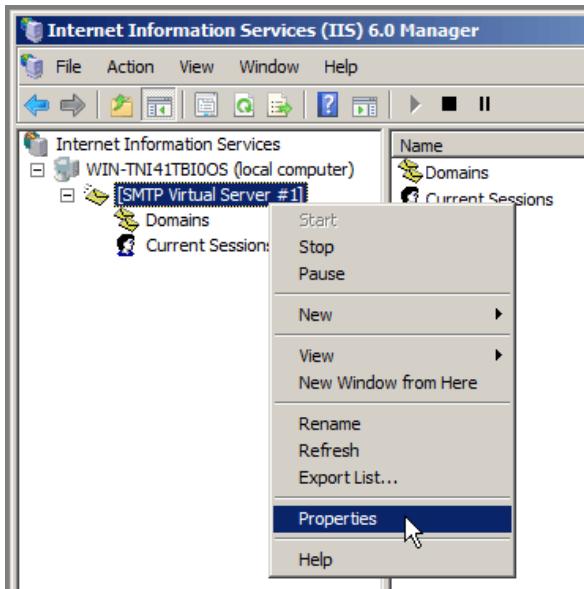
1 <Input iis_ncsa>
2   Module im_file
3   File   'C:\inetpub\logs\LogFiles\W3SVC*\u_nc*.log'
4   <Exec>
5     if $raw_event =~ /(?x)^(\S+)\ -\ (\S+)\ \[(\[\^\]]+)\]\ \"(\S+)\(.\+
6           \ HTTP\/\d.\.\d\"(\S+)\(\S+)
7   {
8     $RemoteHostAddress = $1;
9     if $2 != '-' $UserName = $2;
10    $EventTime = parsedate($3);
11    $HTTPMethod = $4;
12    $HTTPURL = $5;
13    $HTTPResponseStatus = $6;
14    $BytesSent = $7;
15  }
16 </Exec>
17 </Input>
```

57.5. SMTP Server

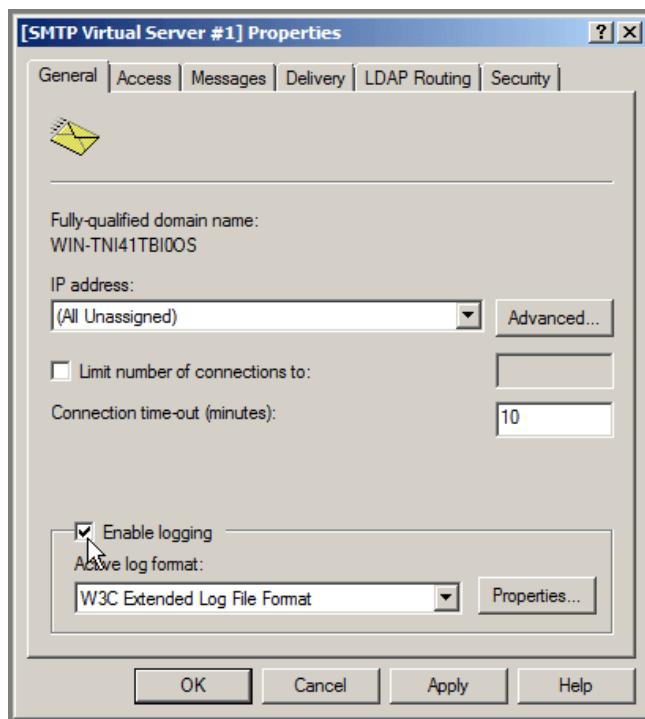
IIS 6.0 in Windows Server 2008 R2 includes an SMTP server. This SMTP server has been deprecated beginning with Windows Server 2012, but it is still available in Windows Server 2016.

IIS SMTP Server logging can be configured as follows.

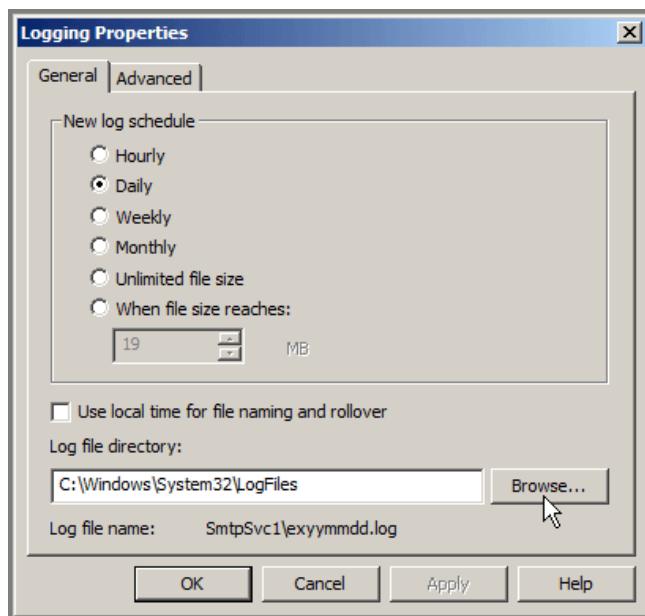
1. Open **Internet Information Services (IIS) 6.0 Manager** from **Administrative Tools**.
2. Right click on the corresponding **SMTP Virtual Server** and click **Properties**.



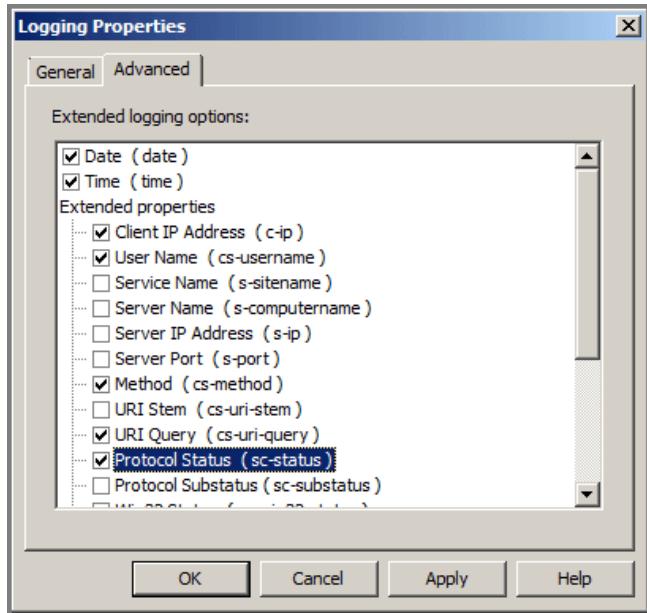
3. Check **Enable logging** and choose the logging format from the **Active log format** drop-down menu. The W3C format is recommended.



4. Click the [Properties...] button to configure the log location and other options.



5. If using the W3C format, adjust the logged fields under the **Advanced** tab. Include the **Date** and **Time** fields and whatever extended properties are required.



Example 244. Collecting W3C Logs From the IIS SMTP Server

The following configuration retrieves W3C logs and parses them using the [xm_w3c](#) module.

nxlog.conf

```

1 <Extension w3c_parser>
2   Module xm_w3c
3 </Extension>
4
5 <Input smtp>
6   Module im_file
7   File 'C:\Windows\System32\LogFiles\SmtpSvc1\ex*.log'
8   InputType w3c_parser
9 </Input>
```

See the preceding sections for more information about processing the other log formats or using [xm_csv](#) for processing W3C logs with NXLog Community Edition.

57.6. Automatic Retrieval of IIS Site Log Locations

The IIS per-site log file locations can be automatically fetched with a batch/PowerShell polyglot script via the [include_stdout](#) directive. For more details, see the PowerShell [Generating Configuration](#) section.

Example 245. Retrieving Log Locations via Script

The following polyglot script should be installed in the NXLog installation (or **ROOT**) directory. It uses the WebAdministration PowerShell module to return the configured log path for each site. If IIS is configured to use one log file per server, the path should instead be configured manually.

WARNING

If there are multiple log formats in the log directory due to configuration changes, the wildcard path should be adjusted to match only those files that are in the corresponding format. For example, for W3C logging use **u_ex*.log** in the last line of the script.

get_iis_log_paths.cmd

```
@( Set "=_= (  
Rem " ) <#  
)  
@Echo Off  
SetLocal EnableExtensions DisableDelayedExpansion  
if defined PROCESSOR_ARCHITEW6432 (  
set powershell=%SystemRoot%\SysNative\WindowsPowerShell\v1.0\powershell.exe  
) else (  
set powershell=powershell.exe  
)  
%powershell% -ExecutionPolicy Bypass -NoProfile ^  
-Command "iex ((gc '%~f0') -join [char]10)"  
EndLocal & Exit /B %ErrorLevel%  
#>  
Import-Module -Name WebAdministration  
foreach($Site in $(get-website)) {  
$LogDir=$($Site.logFile.directory.replace("%SystemDrive%", $env:SystemDrive))  
  
# WARNING: adjust path to match format (for example, for W3C use `u_ex*.log`).  
Write-Output "File '$LogDir\W3SVC $($Site.id)\*.log'" }
```

nxlog.conf

```
1 <Extension w3c_parser>  
2     Module      xm_w3c  
3 </Extension>  
4  
5 <Input iis>  
6     Module      im_file  
7     include_stdout  %ROOT%\get_iis_log_paths.cmd  
8     InputType    w3c_parser  
9 </Input>
```

Chapter 58. Microsoft SharePoint

Microsoft SharePoint Server provides many different types of logs, many of which are configurable. Logs are written to files, databases, and the Windows EventLog. NXLog can be configured to collect these logs, as is shown in the following sections.

See [Monitoring and Reporting in SharePoint Server](#) on TechNet for more information about SharePoint logging.

58.1. Diagnostic Logs

SharePoint diagnostic logs are handled by the Unified Logging Service (ULS), the primary logging mechanism in SharePoint. The ULS writes events to the Windows EventLog and to trace log files. The EventLog and trace log levels of each category or subcategory can be adjusted individually.

The trace log files are generated by and stored locally on each server running SharePoint in the farm, using file names containing the server hostname and timestamp ([HOSTNAME-YYYYMMDD-HHMM.log](#)). SharePoint trace logs are created at regular intervals and whenever there is an IISRESET. It is common for many trace logs to be generated within a 24-hour period.

If configured in the farm settings, each SharePoint server also writes trace logs to the logging database. These logs are written by the **Diagnostic Data Provider: Trace Log** job. NXLog can be configured to collect these logs from the logging database.

For more information about diagnostic logging, see [Configure diagnostic logging in SharePoint Server](#) on TechNet.

58.1.1. ULS Trace Log Format

The Unified Logging Service (ULS) trace log files are tab-delimited.

Trace Log Sample

Timestamp	→ Process	→ TID	→ Area
→ Category	→ EventID → Level	→ Message	→ Correlation
10/12/2017 16:02:18.30	→ hostcontrollerservice.exe (0x0948)	→ 0x191C	→ SharePoint Foundation
→ Topology	→ aup1c → Medium	→ Current app domain:	hostcontrollerservice.exe
(1)←			
10/12/2017 16:02:18.30	→ OWSTIMER.EXE (0x11B8)	→ 0x1AB4	→ SharePoint Foundation
→ Config DB	→ azcxo → Medium	→ SPPersistedObjectCollectionCache: Missed	
memory and file cache, falling back to SQL query. CollectionType=Children,			
ObjectType=Microsoft.SharePoint.Administration.SPWebApplication, CollectionParentId=30801f0f-cca6-			
40bc-9f30-5a4608bbb420, Object Count=1, Stack= at			
Microsoft.SharePoint.Administration.SPPersistedObjectCollectionCache.Get[T](SPPersistedObjectCollect			
ion`1 collection) at			
Microsoft.SharePoint.Administration.SPConfigurationDatabase.Microsoft.SharePoint.Administration.ISPP			
ersistedStoreProvider.GetBackingList[U](SPPersistedObjectCollection`1 persistedCollection) at			
Microsoft.SharePoint.Administration.SPPersistedObjectCollection`1.get_BackcingList() at			
Microsoft.SharePoint.Administration.SPPersistedObjectCollection`1.<GetEnumeratorImpl>d__0.MoveNext()			
at Microsoft.Sh...←			
10/12/2017 16:02:18.30*	→ OWSTIMER.EXE (0x11B8)	→ 0x1AB4	→ SharePoint Foundation
→ Config DB	→ azcxo → Medium	→	
...arePoint.Utilities.SPServerPerformanceInspector.GetLocalWebApplications()		at	
Microsoft.SharePoint.Utilities.SPServerPerformanceInspector..ctor()		at	
Microsoft.SharePoint.Utilities.SPServerPerformanceInspector..cctor()		at	
Microsoft.SharePoint.Administration.SPTimerStore.InitializeTimer(Int64& cacheVersion, Object&			
jobDefinitions, Int32& timerMode, Guid& serverId, Boolean& isServerBusy)		at	
Microsoft.SharePoint.Administration.SPNativeConfigurationProvider.InitializeTimer(Int64&			
cacheVersion, Object& jobDefinitions, Int32& timerMode, Guid& serverId, Boolean& isServerBusy)←			

The ULS log file contains the following fields.

- **Timestamp:** When the event was logged, in local time
- **Process:** Image name of the process logging its activity followed by its process ID (PID) inside parentheses
- **TID:** Thread ID
- **Area:** Component that produced event (SharePoint Portal Server, SharePoint Server Search, etc.)
- **Category:** Detailed category of the event (Topology, Taxonomy, User Profiles, etc.)
- **EventID:** Internal Event ID
- **Level:** Log level of message (Critical, Unexpected, High, etc.)
- **Message:** The message from the application
- **Correlation:** Unique GUID-based ID, generated for each request received by the SharePoint server (unique to each request, not each error)

As shown by the second and third events in the log sample above, long messages span multiple records. In this case, the timestamp of each subsequent record is followed by an asterisk (*). However, trace log messages are not guaranteed to appear consecutively within the trace log. See [Writing to the Trace Log](#) on MSDN.

58.1.2. Configuring Diagnostic Logging

Adjust the log levels, trace log retention policy, and trace log location as follows.

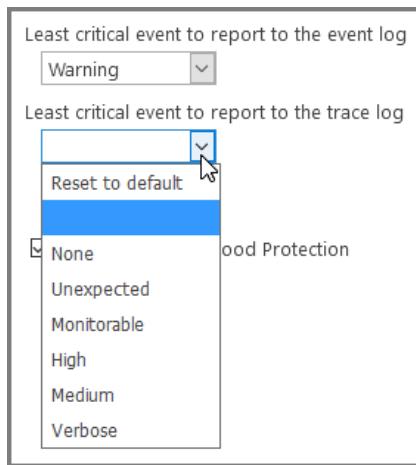
WARNING The diagnostic logging settings are farm-wide.

1. Log in to **Central Administration** and go to **Monitoring > Reporting > Configure diagnostic logging**.
2. In the **Event Throttling** section, use the checkboxes to select a set of categories or subcategories for which to modify the logging level. Expand categories as necessary to view the corresponding subcategories.

Category	Event Level	Trace Level
<input checked="" type="checkbox"/> All Categories		
<input checked="" type="checkbox"/> Access Services		
<input checked="" type="checkbox"/> Administration	Information	Medium
<input type="checkbox"/> Application Design	Information	Medium
<input type="checkbox"/> Application Publishing	Information	Medium
<input type="checkbox"/> Data Layer	Information	Medium
<input type="checkbox"/> Data Pipeline	Information	Medium
<input type="checkbox"/> Database Connection Management	Information	Medium

3. Set the event log and trace log levels for the selected categories or subcategories.

WARNING Only select the verbose level for troubleshooting, as a large number of logs will be generated.



4. To set other levels for other categories or subcategories, click **[OK]** and repeat from step 1.
5. In the **Trace Log** section, adjust the trace log path and retention policy as required. The specified log location must exist on all servers in the farm.

Trace Log

When tracing is enabled you may want the trace log to go to a certain location. Note: The location you specify must exist on all servers in the farm.

Additionally, you may set the maximum number of days to store log files and restrict the maximum amount of storage to use for logging. [Learn about using the trace log](#).

Path	%CommonProgramFiles%\Microsoft Shared\Web Server Extensions\16\LOGS\
Example: %CommonProgramFiles%\Microsoft Shared\Web Server Extensions\16\LOGS\	
Number of days to store log files	14
Restrict Trace Log disk space usage	
<input type="checkbox"/> Restrict Trace Log disk space usage Maximum storage space for Trace Logs (GB) 1000	

6. Click **[OK]** to apply the settings.

Further steps are required to enable writing trace logs to the logging database. For configuring the logging database itself (server, name, and authentication), see the [Configuring Usage Logging](#) section.

1. Log in to **Central Administration** and go to **Monitoring > Timer Jobs > Review job definitions**.
2. Click on the **Diagnostic Data Provider: Trace Log** job.
3. Click the **[Enable]** button to enable the job.
4. Open the **Diagnostic Data Provider: Trace Log** job again and click **[Run Now]** to run the job immediately.

58.1.3. Collecting Diagnostic Logs

The [xm_csv](#) module can be used to parse the tab-delimited trace log files on the local server.

Example 246. Reading the Trace Log Files

This configuration collects logs from the ULS trace log files and uses [xm_csv](#) to parse them. **\$EventTime** and **\$Hostname** fields are added to the event record. Each event is converted to JSON format and written to file.

NOTE

The defined **SHAREPOINT_LOGS** path should be set to the trace log file directory configured in the [Configuring Diagnostic Logging](#) section.

nxlog.conf

```
1 define SHAREPOINT_LOGS C:\Program Files\Common Files\microsoft shared\Web Server \
2 Extensions\16\LOGS
3
4 <Extension json>
5   Module      xm_json
6 </Extension>
7
8 <Extension uls_parser>
9   Module      xm_csv
10  Fields      Timestamp, Process, TID, Area, Category, EventID, Level, Message, \
11      Correlation
12  Delimiter  \t
13 </Extension>
14
15 <Input trace_file>
16   Module      im_file
17   # Use a file mask to read from ULS trace log files only
18   File        '%SHAREPOINT_LOGS%\*-??????-?????.log'
19   <Exec>
20     # Drop header lines and empty lines
21     if $raw_event =~ /(^xEF\xBB\xBF|Timestamp)/ drop();
22   else
23   {
24     # Remove extra spaces
25     $raw_event =~ s/ +(?=\t)//g;
26
27     # Parse with uls_parser instance defined above
28     uls_parser->parse_csv();
29
30     # Set $EventTime field (second precision only)
31     $EventTime = strftime($Timestamp, "%m/%d/%Y %H:%M:%S");
32
33     # Add $Hostname field
34     $Hostname = hostname_fqdn();
35   }
36   </Exec>
37 </Input>
38
39 <Output out>
40   Module    om_file
41   File      'C:\logs\uls.json'
42   Exec      to_json();
43 </Output>
```

Output Sample

```
{
  "EventReceivedTime": "2017-10-12 16:02:20",
  "SourceModuleName": "uls",
  "SourceModuleType": "im_file",
  "Timestamp": "10/12/2017 16:02:18.30",
  "Process": "hostcontrollerservice.exe (0x0948)",
  "TID": "0x191C",
  "Area": "SharePoint Foundation",
  "Category": "Topology",
  "EventID": "aup1c",
  "Level": "Medium",
  "Message": "Current app domain: hostcontrollerservice.exe (1)",
  "EventTime": "2017-10-12 16:02:18",
  "Hostname": "WIN-SHARE.test.com"
}
```

The [im_odbc](#) module can be used to collect diagnostic logs from the farm-wide logging database.

Example 247. Collecting Trace Logs From Database

The following Input configuration collects logs from the **ULSTraceLog** view in the **WSS_UsageApplication** database.

NOTE

The **datetime** data type is not timezone-aware, and the timestamps are stored in UTC. Therefore, an offset is applied when setting the **\$EventTime** field in the configuration below.

nxlog.conf

```

1 <Input trace_db>
2   Module      im_odbc
3   ConnectionString  Driver={ODBC Driver 13 for SQL Server};\
4                                SERVER=SHARESERVE1;DATABASE=WSS_UsageApplication; \
5                                Trusted_Connection=yes
6   IdType      timestamp
7
8   # With ReadFromLast and MaxIdSQL, NXLog will start reading from the last
9   # record when reading from the database for the first time.
10  #ReadFromLast    TRUE
11  #MaxIdSQL       SELECT MAX(LogTime) AS maxid FROM dbo.ULSTraceLog
12
13  SQL      SELECT LogTime AS id, * FROM dbo.ULSTraceLog WHERE LogTime > ?
14  <Exec>
15      # Set $EventTime with correct time zone, remove incorrect fields
16      $EventTime = parsedate(strftime($id, '%Y-%m-%d %H:%M:%S'));
17      delete($id);
18      delete($LogTime);
19  </Exec>
20 </Input>
```

See the [Windows EventLog](#) section below for an example configuration that reads events from the Windows EventLog.

58.2. Usage and Health Data Logs

SharePoint also collects usage and health data to show how it is used. The system generates health and

administrative reports from these logs. Usage and health data logs are written as tab-delimited data to various ***.usage** files in the configured log location path, and also to the logging database.

Log Sample

```
FarmId → UserLogin → SiteSubscriptionId → TimestampUtc → CorrelationId → Action → Target →
Details←
42319181-e881-44f1-b422-d7ab5f8b0117 → TEST\Administrator → 00000000-0000-0000-0000-000000000000 →
2017-10-17 23:15:26.667 → 00000000-0000-0000-0000-000000000000 → Administration.Feature.Install →
AccSrvRestrictedList → {"Id":"a4d4ee2c-a6cb-4191-ab0a-21bb5bde92fb"}←
42319181-e881-44f1-b422-d7ab5f8b0117 → TEST\Administrator → 00000000-0000-0000-0000-000000000000 →
2017-10-17 23:15:26.839 → 00000000-0000-0000-0000-000000000000 → Administration.Feature.Install →
ExpirationWorkflow → {"Id":"c85e5759-f323-4efb-b548-443d2216efb5"}←
```

For more information, see [Overview of monitoring in SharePoint Server](#) on TechNet.

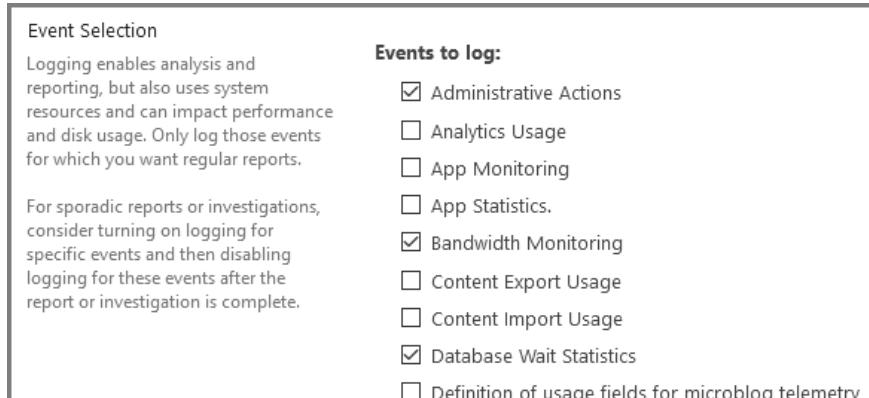
58.2.1. Configuring Usage Logging

Usage and health data collection can be enabled and configured as follows. For more information about configuring usage and health data logging, see [Configure usage and health data collection in SharePoint Server](#) on TechNet.

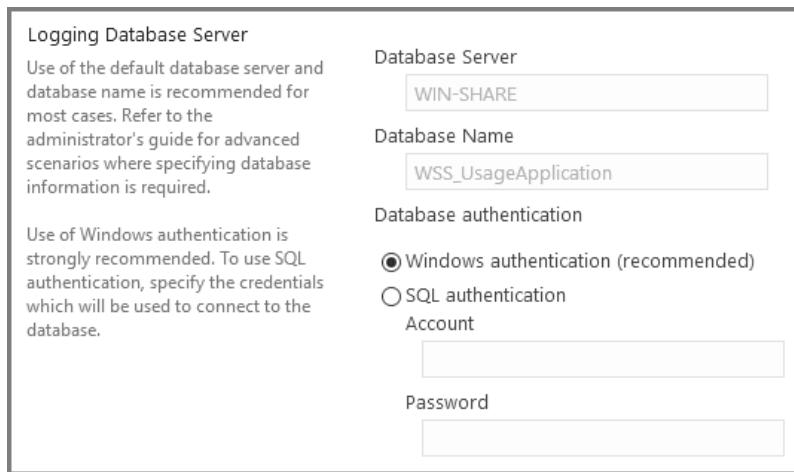
WARNING

The usage and health data collection settings are farm-wide.

1. Log in to **Central Administration** and go to **Monitoring > Reporting > Configure usage and health data collection**.
2. In the **Usage Data Collection** section, check **Enable usage data collection** to enable it.
3. In the **Event Selection** section, use the checkboxes to select the required event categories. It is recommended that only those categories be enabled for which regular reports are required.



4. In the **Usage Data Collection Settings** section, specify the path for the usage log files. The specified log location must exist on all servers in the farm.
5. In the **Health Data Collection** section, check **Enable health data collection** to enable it. Click **Health Logging Schedule** to edit the job definitions for the Microsoft SharePoint Foundation Timer service.
6. Click the **Log Collection Schedule** link to edit the job definitions for the Microsoft SharePoint Foundation Usage service.
7. In the **Logging Database Server** section, adjust the authentication method as required. To change the database server and name, see [Log usage data in a different logging database by using Windows PowerShell](#) on TechNet.



8. Click [OK] to apply the settings.

58.2.2. Collecting Usage Logs

The [xm_csv](#) module can be used to parse the tab-delimited usage and health log files on the local server.

Example 248. Reading Usage Log Files

This configuration collects logs from the **AdministrativeActions** usage log file (see [Using Administrative Actions logging in SharePoint Server 2016](#) on TechNet) and uses [xm_csv](#) to parse them. **\$EventTime** and **\$Hostname** fields are added to the event record. Each event is converted to JSON format and written to file.

NOTE

The defined **SHAREPOINT_LOGS** path should be set to the trace log file directory configured in the [Configuring Diagnostic Logging](#) section.

NOTE

Unlike the diagnostic/trace logs, the various usage/health data categories generate logs with differing field sets. Therefore it is not practical to parse multiple types of usage/health logs with a single [xm_csv](#) parser.

nxlog.conf

```
1 define SHAREPOINT_LOGS C:\Program Files\Common Files\microsoft shared\Web Server \
2 Extensions\16\LOGS
3
4 <Extension json>
5     Module      xm_json
6 </Extension>
7
8 <Extension admin_actions_parser>
9     Module      xm_csv
10    Fields      FarmId, UserLogin, SiteSubscriptionId, TimestampUtc, \
11                  CorrelationId, Action, Target, Details
12    Delimiter   \t
13 </Extension>
14
15 <Input admin_actions_file>
16    Module      im_file
17    # Use a file mask to read from the USAGE files only
18    File        '%SHAREPOINT_LOGS%\AdministrativeActions\*.usage'
19    <Exec>
20        # Drop header lines and empty lines
21        if $raw_event =~ /(^xEF\xBB\xBF|FarmId)/ drop();
22        else
23        {
24            # Parse with parser instance defined above
25            admin_actions_parser->parse_csv();
26
27            # Set $EventTime field
28            $EventTime = parsedate($TimestampUtc + "Z");
29
30            # Add $Hostname field
31            $Hostname = hostname_fqdn();
32        }
33    </Exec>
34 </Input>
35
36 <Output out>
37    Module      om_file
38    File        'C:\logs\uls.json'
39    Exec        to_json();
40 </Output>
```

Output Sample

```
{
  "EventReceivedTime": "2017-10-17 20:46:14",
  "SourceModuleName": "admin_actions",
  "SourceModuleType": "im_file",
  "FarmId": "42319181-e881-44f1-b422-d7ab5f8b0117",
  "UserLogin": "TEST\\Administrator",
  "SiteSubscriptionId": "00000000-0000-0000-0000-000000000000",
  "TimestampUtc": "2017-10-17 23:15:26.667",
  "CorrelationId": "00000000-0000-0000-0000-000000000000",
  "Action": "Administration.Feature.Install",
  "Target": "AccSrvRestrictedList",
  "Details": {
    "Id": "a4d4ee2c-a6cb-4191-ab0a-21bb5bde92fb"
  },
  "EventTime": "2017-10-17 16:15:26",
  "Hostname": "WIN-SHARE.test.com"
}
```

The [im_odb](#) module can be used to collect usage and health logs from the farm-wide logging database.

Example 249. Collecting Usage Logs From Database

The following Input configuration collects Administrative Actions logs from the **AdministrativeActions** view in the **WSS_UsageApplication** database.

NOTE The [datetime](#) data type is not timezone-aware, and the timestamps are stored in UTC. Therefore, an offset is applied when setting the [\\$EventTime](#) field in the configuration below.

nxlog.conf

```

1 <Input admin_actions_db>
2   Module im_odb
3   ConnectionString  Driver={ODBC Driver 13 for SQL Server};\
4                           SERVER=SHARESERVE1;DATABASE=WSS_UsageApplication;\
5                           Trusted_Connection=yes
6   IdType           timestamp
7
8   # With ReadFromLast and MaxIdSQL, NXLog will start reading from the last
9   # record when reading from the database for the first time.
10  #ReadFromLast      TRUE
11  #MaxIdSQL         SELECT MAX(LogTime) AS maxid FROM dbo.AdministrativeActions
12
13  SQL    SELECT LogTime AS id, * FROM dbo.AdministrativeActions WHERE LogTime > ?
14  <Exec>
15    # Set $EventTime with correct time zone, remove incorrect fields
16    $EventTime = parsedate(strftime($id, '%Y-%m-%d %H:%M:%SZ'));
17    delete($id);
18    delete($LogTime);
19  </Exec>
20 </Input>
```

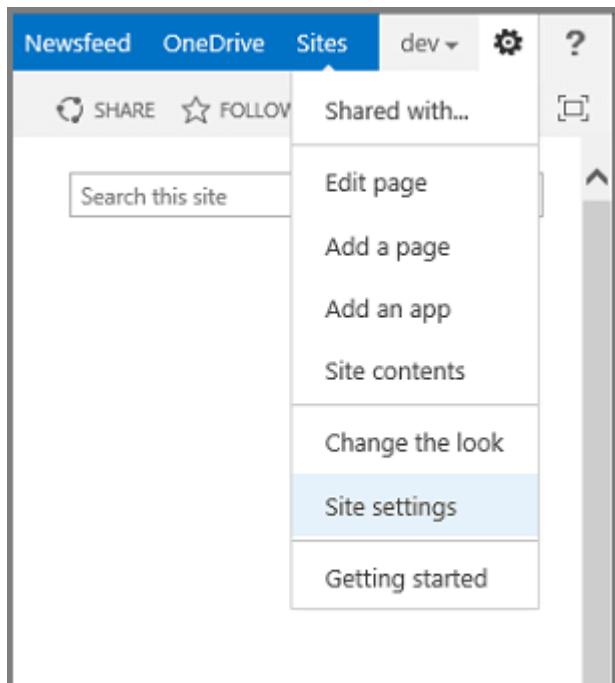
See the [Windows EventLog](#) section for an example configuration that reads events from the Windows EventLog.

58.3. Audit Logs

SharePoint Information Management provides an audit feature that allows tracking of user actions on a site's content. The audit events are stored in the **dbo.AuditData** table in the **WSS_Content** database. The events can be collected via the SharePoint API or by reading the database directly.

Audit logging is disabled by default, and can be enabled on a per-site basis. To enable audit logging, follow these steps. For more details, see the [Configure audit settings for a site collection](#) article on Office Support.

1. Log in to **Central Administration** and go to **Security** > **Information policy** > **Configure Information Management Policy**.
2. Verify that the **Auditing** policy is set to **Available**.
3. On the site collection home page, click **Site actions** (gear icon), then **Site settings**.



4. On the **Site Settings** page, in the **Site Collection Administration** section, click **Site collection audit settings**.

NOTE

If the **Site Collection Administration** section is not shown, make sure you have adequate permissions.

5. Set audit log trimming settings, select the events to audit, and click **[OK]**.

Example 250. Collecting Audit Logs via SharePoint API

This configuration and PowerShell script can be used to collect audit events via SharePoint's API. The script also adds the following fields (performing lookups as required): **\$ItemName**, **\$Message**, **\$SiteURL**, and **\$UserName**. Audit logs are collected from all available sites and the site list is updated each time the logs are collected. See the options in the script header.

NOTE

This configuration may not work correctly when NXLog is used in service mode.

For more details about collecting logs with PowerShell, see the PowerShell [Generating Logs](#) section.

nxlog.conf

```

1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 envvar systemroot
6 <Input audit_powershell>
7   Module im_exec
8   Command "%systemroot%\System32\WindowsPowerShell\v1.0\powershell.exe"
9   Arg    "-ExecutionPolicy"
10  Arg   "Bypass"
11  Arg   "-NoProfile"
12  Arg   "-File"
13  Arg   "C:\auditlog.ps1"
14 <Exec>
15   parse_json();
16   $EventTime = parsedate($EventTime);
17 </Exec>
18 </Input>
```

auditlog.ps1

```

# This script can be used with NXLog to fetch Audit logs via the SharePoint
# API. See the configurable options below. Based on:
# <http://shokochino-sharepointexperience.blogspot.ch/2013/05/create-auditing-reports-in-sharepoint.html>

#Requires -Version 3

# The timestamp is saved to this file for resuming.
$CacheFile = 'C:\nxlog_sharepoint_auditlog_position.txt'

# The database is queried at this interval in seconds.
$PollInterval = 10

# Allow this many seconds for new logs to be written to database.
$ReadDelay = 30

# Use this to enable debug logging (for testing outside of NXLog).
#$DebugPreference = 'Continue'
#####
##### If running 32-bit on a 64-bit system, run 64-bit PowerShell instead.
if ($env:PROCESSOR_ARCHITECTURE -eq "AMD64") {
  Write-Debug "Running 64-bit PowerShell."
  &$env:WINDIR\SysNative\WindowsPowerShell\v1.0\powershell.exe` ` 
  -NonInteractive -NoProfile -ExecutionPolicy Bypass` ` 
  -File "$($myInvocation.InvocationName)" $args
  exit $LASTEXITCODE
}

Add-PSSnapin "Microsoft.SharePoint.PowerShell" -ErrorAction Stop

# Return description for event
function Event-Description {
  param( $entry )
  switch ($entry.Event) {
    AuditMaskChange {"The audit flags are changed for the audited object."}
    ChildDelete {"A child of the audited object is deleted."}
    ChildMove {"A child of the audited object is moved."}
    CheckIn {"A document is checked in."}
```



```

        if($entry.ItemId -eq $List.Id) {
            $ItemName = $List.Title
        }
    }
    # Create hash table
    $record = @{
        # AuditData table fields
        SiteID = $entry.SiteId;
        ItemID = $entry.ItemId;
        ItemType = $entry.ItemType;
        UserID = $entry.UserId;
        AppPrincipalID = $entry.AppPrincipalId;
        MachineName = $entry.MachineName;
        MachineIP = $entry.MachineIP;
        DocLocation = $entry.DocLocation;
        LocationType = $entry.LocationType;
        EventTime = ($entry.Occurred.ToString('o') + "Z");
        Event = $entry.Event;
        EventName = $entry.EventName;
        EventSource = $entry.EventSource;
        SourceName = $entry.SourceName;
       EventData = $entry.EventData;
        # Additional fields
        ItemName = $ItemName;
        Message = Event-Description $entry;
        SiteURL = $site.Url;
        UserName = $UserName;
    }
    # Return record as JSON
    $record | ConvertTo-Json -Compress | Write-Output
}
}

# Get position timestamp from cache file. On first run, create file using
# current time.
function Get-Position {
    param( $file )
    Try {
        if (Test-Path $file) {
            $timestamp = (Get-Date (Get-Content $file -First 1))
            $timestamp = $timestamp.ToUniversalTime()
            $start = $timestamp.AddTicks(-($timestamp.Ticks % 10000000))
        }
        else {
            $timestamp = [System.DateTime]::UtcNow
            $timestamp = $timestamp.AddTicks(-($timestamp.Ticks % 10000000))
            Save-Position $file $timestamp
        }
        return $start
    }
    Catch {
        Write-Error "Failed to read timestamp from position file."
        exit 1
    }
}

# Save position timestamp to cache file.
function Save-Position {
    param( $file, $time )
    Try { Out-File -FilePath $file -InputObject $time.ToString('o') }
}

```

```
Catch {
    Write-Error "Failed to write timestamp to position file."
    exit 1
}

# Main
Try {
    $start = Get-Position $CacheFile
    Write-Debug "Got start time of $($start.ToString('o'))."
    $now = [System.DateTime]::UtcNow
    $now = $now.AddTicks(-($now.Ticks % 10000000))
    Write-Debug "Got current time of $($now.ToString('o'))."
    $diff = ($now - $start).TotalSeconds
    # Check whether waiting is required to comply with $ReadDelay.
    if (($diff - $PollInterval) -lt $ReadDelay) {
        $wait = $ReadDelay - $diff + $PollInterval
        Write-Debug "Waiting $wait seconds to start collecting logs."
        Start-Sleep -Seconds $wait
    }
    # Repeatedly read from the audit log
    while($true) {
        Write-Debug "Using range start time of $($start.ToString('o'))."
        $now = [System.DateTime]::UtcNow
        $now = $now.AddTicks(-($now.Ticks % 10000000))
        $end = $now.AddSeconds(-($ReadDelay))
        Write-Debug "Using range end time of $($end.ToString('o'))."
        $sites = Get-SPSite -Limit All
        foreach($site in $sites) { Get-Audit-Data $site $start $end }
        Write-Debug "Saving position timestamp to cache file."
        Save-Position $CacheFile $end
        Write-Debug "Waiting $PollInterval seconds before reading again."
        Start-Sleep -Seconds $PollInterval
        $start = $end
    }
}
Catch {
    Write-Error "An unhandled exception occurred!"
    exit 1
}
```

Output Sample

```
{
  "EventReceivedTime": "2018-03-01 02:12:45",
  "SourceModuleName": "audit_ps",
  "SourceModuleType": "im_exec",
  "UserID": 18,
  "LocationType": 0,
  "EventName": null,
  "MachineName": null,
  "ItemName": null,
  "EventData": "<Version><AllVersions/></Version><Recycle>1</Recycle>",
  "Event": 4,
  "UserName": "i:0#.w|test\\test",
  "SourceName": null,
  "SiteURL": "http://win-share",
  "EventTime": "2018-03-01 02:12:12",
  "EventSource": 0,
  "Message": "The audited object is deleted.",
  "DocLocation": "Shared Documents/document.txt",
  "ItemID": "48341996-7844-4842-bef6-94b43ace0582",
  "SiteID": "51108732-0903-4721-aae7-0f9fb5aebfc2",
  "MachineIP": null,
  "AppPrincipalID": 0,
  "ItemType": 1
}
```

Example 251. Collecting Audit Logs Directly From Database

This configuration collects audit events from the **AuditData** table in the **WSS_Content** database.

NOTE

The **datetime** data type is not timezone-aware, and the timestamps are stored in UTC. Therefore, an offset is applied when setting the **\$EventTime** field in the configuration below.

nxlog.conf

```

1 <Input audit_db>
2   Module           im_odbc
3   ConnectionString Driver={ODBC Driver 13 for SQL Server}; \
4                                Server=SHARESERVE1; Database=WSS_Content; \
5                                Trusted_Connection=yes
6   IdType          timestamp
7
8   # With ReadFromLast and MaxIdSQL, NXLog will start reading from the last
9   # record when reading from the database for the first time.
10  #ReadFromLast      TRUE
11  #MaxIdSQL        SELECT MAX(Occurred) AS maxid FROM dbo.AuditData
12
13  SQL              SELECT Occurred AS id, * FROM dbo.AuditData WHERE Occurred > ?
14  <Exec>
15    # Set $EventTime with correct time zone, remove incorrect fields
16    $EventTime = parsedate(strftime($id, '%Y-%m-%d %H:%M:%S'));
17    delete($id);
18    delete($occurred);
19  </Exec>
20 </Input>
```

58.4. Windows EventLog

SharePoint will generate Windows event logs according to the diagnostic log levels configured (see the [Diagnostic Logs](#) section). NXLog can be configured to collect logs from the Windows EventLog as shown below. For more information about collecting Windows EventLog events with NXLog, see the [Windows EventLog](#) chapter.

Example 252. Collecting SharePoint Events From the EventLog

This configuration uses the `im_msvistalog` module to collect all logs from four SharePoint crimson channels, as well as Application channel events of Warning or higher level. The Application channel will include other non-SharePoint events. There may be other SharePoint events generated which will not be collected with this query, depending on the configuration and the channels used.

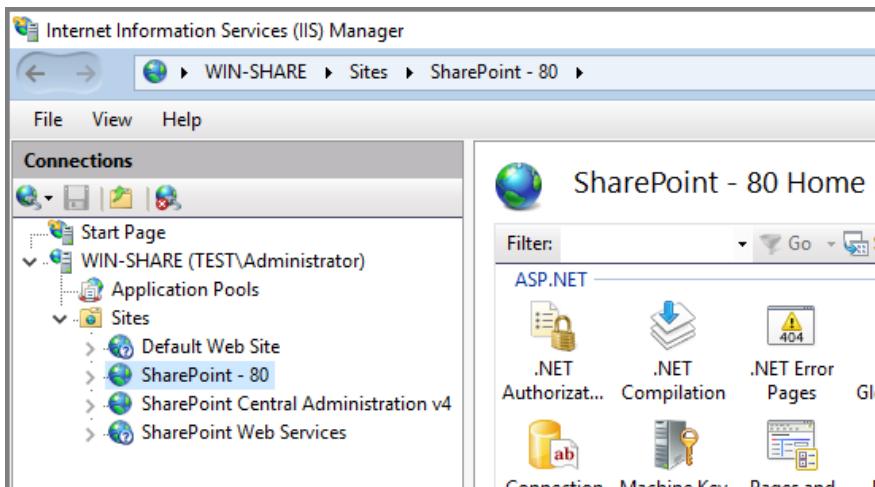
`nxlog.conf`

```

1 <Input eventlog>
2   Module im_msvistalog
3   <QueryXML>
4     <QueryList>
5       <Query Id="0" Path="Application">
6         <Select Path="Application">
7           *[System[(Level=1 or Level=2 or Level=3)]]</Select>
8         <Select Path="System">
9           *[System[(Level=1 or Level=2 or Level=3)]]</Select>
10        <Select Path="Microsoft-Office Server-Search/Operational">
11          *</Select>
12        <Select Path="Microsoft-Office-EduServer Diagnostics">&lt;/Select>
13        <Select Path="Microsoft-SharePoint Products-Shared/Operational">
14          *</Select>
15        <Select Path="Microsoft-SharePoint Products-Shared/Audit">&lt;/Select>
16      </Query>
17    </QueryList>
18  </QueryXML>
19 </Input>
```

58.5. IIS Logs

SharePoint uses the Internet Information Server (IIS) to serve the configured sites as well as the Central Administration site. IIS generates its own logs.



See the [Microsoft IIS](#) chapter for more information about collecting events from IIS with NXLog.

Chapter 59. Microsoft SQL Server

NXLog can be configured to collect both error logs and audit logs from Microsoft SQL Server.

59.1. Error Log

Microsoft SQL Server stores its logs in UTF-16 encoding using a line-based format. It is recommended to normalize the encoding to UTF-8 as shown in the example below.

Example 253. Local Collection from the Microsoft SQL Server Error Log

This example collects logs from the Microsoft SQL Server error log, and uses the `xm_charconv` module [convert\(\)](#) function to convert the character set to UTF-8.

`nxlog.conf`

```
1 <Extension _charconv>
2     Module xm_charconv
3 </Extension>
4
5 <Input in>
6     Module im_file
7     File   "C:\\MSSQL\\ERRORLOG"
8     <Exec>
9         $raw_event = convert($raw_event, 'UCS-2LE','UTF-8');
10        if $raw_event == '' drop();
11    </Exec>
12 </Input>
```

NOTE

As of this writing, the LineBased parser, the default InputType for `im_file`, is not able to properly read the double-byte UTF-16 encoded files and will read an additional empty line (because of the double-byte CRLF). The above `drop()` call is intended to fix this. Also, `convert_fields('UTF-16', 'UTF-8');` might work instead of UCS-2LE.

59.2. Audit Log

Microsoft SQL Server 2008 introduced a new feature that provided a much needed solution for security oriented customers: SQL Server Auditing. With this feature, the server records all changes to the database and access groups. These logs are stored in a proprietary format file or in the Application/Security EventLog data.

While in earlier versions these logs had to be generated by SQL Trace or a custom monitoring process, it is now possible to start recording audit logs with a few clicks in Management Studio or a relatively simple SQL script.

The following instructions require a Microsoft SQL Server with auditing support and the Microsoft SQL Management Studio. Consult the relevant documentation below to determine whether "Fine Grained Auditing" is available for your SQL Server version and edition.

- [Features Supported by the Editions of SQL Server 2008 R2](#)
- [Features Supported by the Editions of SQL Server 2012](#)
- [Features Supported by the Editions of SQL Server 2014](#)
- [Editions and Supported Features for SQL Server 2016](#)

59.2.1. Configuring SQL Server for Auditing

To set up SQL auditing, you will need to create a **Server Audit** object that describes the target for audit data (a

binary file or EventLog channel). Then you will need either a **Server Audit Specification** object or a **Database Audit Specification** object (or both) so SQL Server can start producing meaningful data into the Server Audit object you defined.

Generally, if you are interested in logging SQL statements you should set up a Database Audit Specification object. If you need to trace server events like logon attempts or server principle changes, you should define a Server Audit Specification.

The creation of these objects are straightforward, but must be executed correctly to be able to work together with NXLog. For more information on the subject, and a detailed walkthrough of the SQL Server side, read the Microsoft TechNet [Auditing in SQL Server 2008](#) article.

59.2.1.1. Creating a Server Audit Object

GUI

In Management Studio, after connecting to your database instance:

1. Click on the plus (+) next to **Security**.
2. Right-click on **Audits** and select **New Audit**. The **Create Audit** dialog box appears. Choose a name for the audit object.
3. In the **Audit destination** drop-down list, choose **Security log** or **File** (for security reasons, **Application log** is not recommended as a target). If you chose **File**, enter a file path and configure log rotation.
4. Click **OK**. The Server Audit object is created.

Note the red arrow next to the newly created object's name indicating this is a disabled object. To enable it, right click on the audit object and select **Enable audit**.

SQL script

Instead of creating the Server Audit object via the GUI, you can alternatively run the **CREATE SERVER AUDIT** and **ALTER SERVER AUDIT** commands. For example:

```
CREATE SERVER AUDIT myaudit
TO <SECURITY LOG|FILE>
WITH (QUEUE_DELAY=100, ON_FAILURE=CONTINUE);
ALTER SERVER AUDIT myaudit WITH (STATE=ON)
```

59.2.1.2. Creating a Server Audit Specification

GUI

In Management Studio, after connecting to your database instance:

1. Click on the plus (+) next to **Security**.
2. Right-click on **Server Audit Specifications** and select **New Audit**. The **Create Audit** dialog box appears.
3. Choose a Server Audit object (the one you defined earlier) and select the actions you want to get an audit report of.
4. Click **OK**. The Server Audit Specification object is created.

Note the red arrow next to the newly created object's name indicating this is a disabled object. To enable it, right click on the audit object and select **Enable audit**.

SQL script

Alternatively, you can use the **CREATE SERVER AUDIT SPECIFICATION** and **ALTER SERVER AUDIT SPECIFICATION** commands. For example:

```

CREATE SERVER AUDIT SPECIFICATION srv_audit_spec
FOR SERVER AUDIT myaudit
    ADD (FAILED_LOGIN_GROUP)
ALTER SERVER AUDIT SPECIFICATION srv_audit_spec
FOR SERVER AUDIT myaudit
WITH (STATE=ON)

```

59.2.1.3. Creating a Database Audit Specification

GUI

In Management Studio, after connecting to your database instance:

1. Click on the plus (+) next to **Databases**.
2. Click on the plus (+) next to the database you want to audit, then click on the plus (+) next to **Security** under your database.
3. Right-click on **Database Audit Specifications** and select **New Audit**. The **Create Audit** dialog box appears.
4. Choose a Server Audit object (the one you defined earlier) and select the actions you want to get an audit report of.
5. Click **OK**. The Database Audit Specification object is created.

SQL script

Alternatively, you can use the **CREATE DATABASE AUDIT SPECIFICATION** and **ALTER DATABASE AUDIT SPECIFICATION** commands. For example:

```

CREATE DATABASE AUDIT SPECIFICATION mydb_audit_spec
FOR SERVER AUDIT myaudit
ADD (SELECT ON OBJECT::[Production].[Product] BY [Peter])
ALTER DATABASE AUDIT SPECIFICATION mydb_audit_spec
FOR SERVER AUDIT myaudit
    ADD (SELECT
        ON OBJECT::dbo.Table1
        by dbo)
WITH (STATE = ON);

```

59.2.2. Checking SQL Audit Generation

Audit file

If you chose **File** as the audit target, check if the file is created and grows when the audit criteria are met. Other than incorrect NTFS permissions, there should not be any issue with this type of log target.

EventLog

Check your Security and Application EventLogs to see if SQL Auditing is working properly. If there are no related events in the Security log, even though you set that as destination, check the Application log too. You will find event ID **33204** in your Application log indicating SQL Server's failure to write to the security log.

This is a registry related permission error: the account running your SQL server instance is unable to create an entry under **HKLM\SYSTEM\CurrentControlSet\Services\EventLog\Security** and fails with ID **33204**.

This error can be fixed as follows.

1. Run **regedit**.
2. Grant **Full Control** permission for the account running your SQL server instance (for example, Network Service or a named account) to **HKLM\SYSTEM\CurrentControlSet\Services\EventLog\Security**.

3. Disable, then re-enable your Server Audit; this creates a sub-key, **MSSQLSERVER\$AUDIT**.
4. Optionally, remove the **Full Control** permission you just added. This permission is no longer required now that the sub-key has been created.

59.2.3. Configuring Collection of SQL Audit Logs

After a working SQL audit is in place, NXLog can be configured to read the logs from the Server Audit object. If it is configured with a **Security log** destination, the events can be read from the EventLog. If it is configured with a **File** target, the events can be queried via ODBC.

59.2.3.1. Reading From Windows EventLog

The [im_msvisalog](#) module can be used to read events from the EventLog.

Example 254. Reading SQL Server Audit Events From the EventLog

In this example, events with ID 33205 are retrieved, **\$Message** is parsed and additional fields are set, and finally the event is converted to JSON format.

Sample Event

```
2011-11-11 11:00:00 sql2008-ent AUDIT_SUCCESS 33205 Audit event: event_time:2011-11-11
11:00:00.000000000
sequence_number:1
action_id:SL
succeeded:true
permission_bitmask:1
is_column_permission:true
session_id:57
server_principal_id:264
database_principal_id:1
target_server_principal_id:0
target_database_principal_id:0
object_id:2105058535
class_type:U
session_server_principal_name:SQL2008-ENT\myuser
server_principal_name:SQL2008-ENT\myuser
server_principal_sid:01050000000000212000001aaaaaabbbcccccdddeeeefffffff
database_principal_name:dbo
target_server_principal_name:*
target_server_principal_sid:*
target_database_principal_name:*
server_instance_name:SQL2008-ENT
database_name:logindb
schema_name:dbo
object_name:users
statement:select username nev from dbo.users;
additional_information:
```

nxlog.conf

```
1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 <Input in>
6   Module im_msvisalog
7   <QueryXML>
8     <QueryList>
9       <Query Id="0" Path="Security">
10         <Select Path="Security">*[System[ (EventID=33205) ]]</Select>
11       </Query>
12     </QueryList>
13   </QueryXML>
14   <Exec>
15     if $Message =~ /action_id:(.*)/ $ActionId = $1;
16     if $Message =~ /session_server_principal_name:(.*)/ $SessionSPN = $1;
17     if $Message =~ /database_principal_name:(.*)/ $DBPrincipal = $1;
18     if $Message =~ /server_instance_name:(.*)/ $ServerInstance = $1;
19     if $Message =~ /database_name:(.*)/ $DBName = $1;
20     if $Message =~ /schema_name:dbo(.*)/ $SchemaName = $1;
21     if $Message =~ /object_name:(.*)/ $ObjectName = $1;
22     if $Message =~ /statement:(.*)/ $Statement = $1;
23     to_json();
24   </Exec>
25 </Input>
```

59.2.3.2. Reading From the Audit File

The audit file is a special, proprietary format file that can be accessed with SQL commands. In order to query it, an ODBC connection must be set up between NXLog and the SQL server. For NXLog on Windows, set up a 32-bit ODBC datasource using the SQL Server Native Client driver by running the 32-bit ODBC Data Source Administrator (`%WINDIR%\SysWOW64\odbcad32.exe`). Note that it is mandatory to use the 32-bit version of the ODBC administrator. For NXLog on Linux, consult the [unixODBC documentation](#) for more information about setting up an ODBC data source.

After the ODBC connection has been set up and successfully tested, configure NXLog with an Input block that uses the `im_odbc` module. Enter a SELECT query that reads from the audit file (via the `fn_get_audit_file` function) and sorts events by the audit event time cast as an integer timestamp with the field name `id`. Producing this `id` field may not be trivial, see the example below. The SELECT statement must include a WHERE clause (see the `im_odbc` [SQL](#) directive), otherwise `im_odbc` will read logs in an endless loop. Also note that `IdIsTimeStamp` must be set to FALSE (the `id` column is not a real timestamp).

Example 255. Reading SQL Server Audit Events From SQL Audit File

The following Input block uses the `im_odbc` module to collect logs via ODBC. Several additional fields are set from the `$Message` field.

nxlog.conf

```

1 <Input in>
2   Module          im_odbc
3   ConnectionString DSN=AuditsSQL;uid=myuser;pwd=secretpassword;database=master
4   SQL SELECT CAST(DATEDIFF(minute, '19700101', CAST(AF.event_time AS DATE)) AS \
5     BIGINT) * 60000 + DATEDIFF(ms, '19000101', CAST(AF.event_time AS TIME)) AS \
6     'id',* FROM fn_get_audit_file('C:\auditlogs\myaudit_*.sqlaudit', default, \
7     default) AF WHERE CAST(DATEDIFF(minute, '19700101', CAST(AF.event_time AS \
8     DATE)) AS BIGINT) * 60000 + DATEDIFF(ms, '19000101', CAST(AF.event_time AS \
9     TIME)) > ?
10  PollInterval    5
11  IdIsTimeStamp  FALSE
12  <Exec>
13    if $Message =~ /action_id:(.*)/ $ActionId = $1;
14    if $Message =~ /session_server_principal_name:(.*)/ $SessionSPN = $1;
15    if $Message =~ /database_principal_name:(.*)/ $DBPrincipal = $1;
16    if $Message =~ /server_instance_name:(.*)/ $ServerInstance = $1;
17    if $Message =~ /database_name:(.*)/ $DBName = $1;
18    if $Message =~ /schema_name:dbo(.*)/ $SchemaName = $1;
19    if $Message =~ /object_name:(.*)/ $ObjectName = $1;
20    if $Message =~ /statement:(.*)/ $Statement = $1;
21  </Exec>
22 </Input>
```

59.2.3.3. Translating the "action_id" to a Meaningful Action

The `action_id` in the received events contains the operation that had to be logged. Unfortunately this field is a bit obscure. Either consult the statement field for an explanation or create a match between the `action_id` and its human readable version. To get a list of audit actions with the matching `action_id`, execute this query and save the table for further reference.

```
SELECT DISTINCT action_id, name, class_desc, parent_class_desc
FROM sys.dm_audit_actions
```

Chapter 60. Microsoft System Center Operations Manager

Microsoft System Center Operations Manager (SCOM) provides infrastructure monitoring across various services, devices, and operations from a single console. The activities related to these systems are recorded in SCOM's databases, and these databases can be queried using SQL. The resulting data can be collected and forwarded by NXLog.

60.1. Log Types

Collected event logs

These events are collected by filtering rules in configured management packs.

Alert logs

Alerts are significant events generated by rules and monitors.

SCOM administrative event logs

Administrative actions executed in SCOM are currently either unsupported by Microsoft (requiring SQL triggers in the OM database and thus voiding the warranty) or are too performance heavy with little meaningful data to retrieve.

NOTE The default retention time for resolved alerts and collected events is seven days, after which the database entries are groomed. To configure database grooming settings read the TechNet article [How to Configure Grooming Settings for the Operations Manager Database](#).

60.2. Collecting Logs

For NXLog to collect logs, the following prerequisites must be completed.

- Create a Windows/SQL account with read permissions for the Operations Manager database.
- Configure an ODBC 32-bit System Data Source on the server running NXLog. For more information, consult the relevant ODBC documentation: the [Microsoft ODBC Data Source Administrator guide](#) or the [unixODBC Project](#).
- Set an appropriate firewall rule on the database server that accepts connections from the server running NXLog. Open TCP port 1433 or whichever port the SQL Server is configured to allow SQL Server access on. For further information read the [Configure Firewall for Database Engine Access guide](#).

NXLog can then be configured with one or more `im_odbc` input modules, each with an `SQL` query that produces the fields to be logged.

NOTE The configured `SQL` query must contain a way to serialize the result set, enabling NXLog to resume reading logs where it left off after a restart. This is easily achieved by using an auto-increment-like solution or a timestamp field. See the example below.

Example 256. Collecting Event and Alert Logs

This example queries the database for event logs and unresolved alert logs, then sends the results in JSON format to a plain text file. Note the `Exec` directive in the `scom_alerts` input instance. It is used to extract the content of the `AlertParameters` field that is itself a composite (XML) structure. You should define your own regular expressions to extract data you are interested in from the alerts' `AlertParameters` and `Context` fields and the events' `EventData` and `EventParameters` fields.

This example uses the `DATEDIFF` SQL function to generate a timestamp from an SQL datetime field with millisecond precision. The timestamp is used to serialize the result set as required by NXLog. Starting with

SQL Server 2016 the DATEDIFF_BIG T-SQL function can be used instead (see [DATEDIFF_BIG \(Transact-SQL\)](#) at MSDN).

nxlog.conf

```

1 <Extension xm_json>
2   Module      xm_json
3 </Extension>
4
5 <Input scom_events>
6   Module      im_odbc
7   ConnectionString  DSN=scom;uid=username@mydomain.local;pwd=mypassword; \
8                               database=OperationsManager
9   SQL SELECT CAST(DATEDIFF(minute, '19700101', CAST(EV.TimeGenerated AS DATE)) \ 
10          AS BIGINT) * 60000 + DATEDIFF(ms, '19000101', \
11          CAST(EV.TimeGenerated AS TIME)) AS 'id', \
12          EV.TimeGenerated AS 'EventTime', \
13          EV.TimeAdded AS 'EventAddedTime', \
14          EV.Number AS 'EventID', \
15          EV.MonitoringObjectDisplayName AS 'Source', \
16          R.DisplayName AS 'RuleName', \
17          EV.EventData, EV.EventParameters \
18   FROM EventView EV JOIN RuleView R WITH (NOLOCK) ON \
19          EV.RuleId = R.id \
20   WHERE CAST(DATEDIFF(minute, '19700101', CAST(EV.TimeGenerated \ 
21          AS DATE)) AS BIGINT) * 60000 + DATEDIFF(ms, '19000101', \
22          CAST(EV.TimeGenerated AS TIME)) > ?
23   PollInterval    30
24   IdIsTimeStamp  FALSE
25 </Input>
26
27 <Input scom_alerts>
28   Module      im_odbc
29   ConnectionString  DSN=scom;uid=username@mydomain.local;pwd=mypassword; \
30                               database=OperationsManager
31   SQL SELECT CAST(DATEDIFF(minute, '19700101', CAST(AL.TimeRaised AS DATE)) AS \
32          BIGINT) * 60000 + DATEDIFF(ms, '19000101', \
33          CAST(AL.TimeRaised AS TIME)) AS 'id', \
34          AL.AlertStringName AS 'AlertName', \
35          AL.Category AS 'Category', \
36          AL.AlertStringDescription AS 'AlertDescription', \
37          AL.TimeRaised AS 'EventTime', \
38          AL.TimeAdded AS 'EventAddedTime', AL.Context, \
39          AL.AlertParams AS 'AlertParameters' \
40   FROM AlertView AL \
41   WHERE AL.resolutionstate <> 255 AND CAST(DATEDIFF(minute, \
42          '19700101', CAST(AL.TimeRaised AS DATE)) AS BIGINT) * \
43          60000 + DATEDIFF(ms, '19000101', CAST(AL.TimeRaised AS \
44          TIME)) > ?
45 <Exec>
46   if $AlertParameters =~ /(?x)\<AlertParameters\>\<AlertParameter\d\>(.*) \
47           \<\/AlertParameter\d\>\<\/AlertParameters\>$/sm
48   $AlertMessage = $1;
49 </Exec>
50   PollInterval    30
51   IdIsTimeStamp  FALSE
52 </Input>
53
54 <Output outfile>
55   Module      om_file
56   File        'C:\logs\out.log'

```

```
57     Exec          to_json();  
58 </Output>  
59  
60 <Route r>  
61     Path         scom_events, scom_alerts => outfile  
62 </Route>
```

Output Sample (Event Log)

```
{  
    "id": 1463035663720,  
    "EventTime": "2016-05-12 06:47:43",  
    "EventAddedTime": "2016-05-12 06:48:15",  
    "WindowsID": 4776,  
    "Source": "dc01.nxlog.local",  
    "RuleName": "Windows log collection test",  
    "EventData": "<DataItem type=\"System.XmlData\" time=\"2016-05-12T08:47:44.7224395+02:00\" sourceHealthServiceId=\"F767895D-A408-0F91-42A3-87565E1D9D85\"><EventData xmlns=\"http://schemas.microsoft.com/win/2004/08/events/event\"><Data Name=\"PackageName\">MICROSOFT_AUTHENTICATION_PACKAGE_V1_0</Data><Data Name=\"TargetUserName\">SCOM01$</Data><Data Name=\"Workstation\">SCOM01</Data><Data Name=\"Status\">0x0</Data></EventData></DataItem>",  
    "EventParameters":  
        "<Param>MICROSOFT_AUTHENTICATION_PACKAGE_V1_0</Param><Param>SCOM01$</Param><Param>SCOM01</Param><Param>0x0</Param>",  
    "EventReceivedTime": "2016-05-12 10:28:50",  
    "SourceModuleName": "in_scom_events",  
    "SourceModuleType": "im_odbc"  
}
```

Output Sample (Alert Log)

```
{  
    "id": 1462887688220,  
    "Alert Name": "Failed to Connect to Computer",  
    "Category": "StateCollection",  
    "Alert Description": "The computer {0} was not accessible.",  
    "EventTime": "2016-05-10 13:41:28",  
    "EventAddedTime": "2016-05-10 13:41:28",  
    "Context": "<DataItem type=\"MonitorTaskDataType\" time=\"2016-05-10T15:41:28.1932994+02:00\" sourceHealthServiceId=\"00000000-0000-0000-0000-000000000000\"><StateChange><DataItem time=\"2016-05-10T15:41:25.5592943+02:00\" type=\"System.Health.MonitorStateChangeData\" sourceHealthServiceId=\"D53BAD42-4C93-6634-E610-BDC3E38ABD5B\" MonitorExists=\"true\" DependencyInstanceId=\"00000000-0000-0000-0000-000000000000\" DependencyMonitorId=\"00000000-0000-0000-0000-000000000000\"><ManagedEntityId>CC7109D1-9177-090D-AC3A-18781CFFF898</ManagedEntityId><EventOriginId>9B02AB65-FDB5-40AE-863F-6FAD232E06F9</EventOriginId><MonitorId>B59F78CE-C42A-8995-F099-E705DBB34FD4</MonitorId><ParentMonitorId>A6C69968-61AA-A6B9-DB6E-83A0DA6110EA</ParentMonitorId><HealthState>3</HealthState><OldHealthState>1</OldHealthState><TimeChanged>2016-05-10T15:41:25.5592943+02:00</TimeChanged><Context><DataItem type=\"System.Availability.StateData\" time=\"2016-05-10T15:41:25.5542835+02:00\" sourceHealthServiceId=\"D53BAD42-4C93-6634-E610-BDC3E38ABD5B\"><ManagementGroupId>{1457194C-D3B4-6685-5D3B-E4F7DAB158AD}</ManagementGroupId><HealthServiceId>72704AC7-4FDF-6006-1BB0-C74868E173D5</HealthServiceId><HostName>member2012r2-01.nxlog.local</HostName><Reachability>ThruServer=false</Reachability><State>0</State><Diagnostic><DataItem type=\"System.PropertyBagData\" time=\"2016-05-10T15:41:25.6342865+02:00\" sourceHealthServiceId=\"D53BAD42-4C93-6634-E610-BDC3E38ABD5B\"><Property Name=\"StatusCode\" VariantType=\"8\">11003</Property><Property Name=\"ResponseTime\" VariantType=\"8\"></Property></DataItem></Diagnostic></DataItem>, <AlertParameters> <AlertParameters><AlertParameter1>member2012r2-01.nxlog.local</AlertParameter1></AlertParameters> ",  
    "EventReceivedTime": "2016-05-12 10:33:38",  
    "SourceModuleName": "in_scom_alerts",  
    "SourceModuleType": "im_odbc",  
    "AlertMessage": "member2012r2-01.nxlog.local"  
}
```

Chapter 61. MongoDB

NXLog can be configured to collect data from a MongoDB database.

Example 257. Collecting From MongoDB

In this example, the `im_perl` module is used to execute a Perl script which pulls the data from the MongoDB database.

`nxlog.conf`

```
1 <Input mongodb>
2   Module    im_perl
3   PerlCode  /opt/nxlog/bin/mongodb-input.pl
4 </Input>
```

When new documents are available in the database, the script will sort them by ObjectId, process them sequentially, and pass them to NXLog by calling `Log::Nxlog::add_input_data()`. Finally, the document is deleted from the collection. The script will poll the database continuously by setting a timer with `Log::Nxlog::set_read_timer()`. In the event that the MongoDB server is unreachable (and an exception is thrown while trying to manipulate the data), the timer is set to a different value so that the connection can be reestablished at a later time.

WARNING

Please keep in mind that the script deletes files from MongoDB after download. This might not be the desired behavior for your particular scenario. For instance, the original files may be need to be preserved for compliance or forensic reasons.

`mongodb-input.pl`

```
#!/usr/bin/perl

use strict;
use warnings;

use FindBin;
use lib $FindBin::Bin;
use Log::Nxlog;
use MongoDB;
use Try::Tiny;

my $counter;
my $client;
my $collection;
my $cur;
my $count;
my $logfile;

sub read_data_int
{
    $counter //+= 1;
    # Connect to the server
    $client //=> MongoDB::MongoClient->new(host => 'localhost:27017');

    # Select the database and collection
    $collection //=> $client->ns('zips.zips');
    # Sort all existing documents by _id.
    $cur = $collection->find()->sort({_id => 1});

    # Do this only the first time around. Make our cursor immortal.
    if ($counter == 1) {
```

```
$cur->immortal(1);
}

$counter++;

# If any new document exist, process them.
while ($cur->has_next()) {
    my $event = Log::Nxlog::logdata_new();
    my $obj = $cur->next;
    my $line = "ID: " . $obj->{"_id"} . " City: " . $obj->{"city"} . " Loc: " . $obj->{"loc"}[0] . "," . $obj->{"loc"}[1] . " Pop: " . $obj->{"pop"} . " State: " . $obj->{"state"};
    Log::Nxlog::set_field_string($event, 'raw_event', $line);
    Log::Nxlog::add_input_data($event);
    # Once the document is processed, delete it.
    my $result = $collection->delete_one( { '_id' => $obj->{"_id"} } );
    #print $logfile "Extracted document with _id: " . $obj->{"_id"} . " Deleting returned: "
    . $result->deleted_count . "\n";
}

sub read_data
{
    # Use a try/catch block in order to resume when mongodb is unreachable.
    #open ($logfile, '>>', '/tmp/perl-input.log') or die "Could not open log file";
    try {
        read_data_int();
        # Adjust this timer for how often to look for new documents.
        Log::Nxlog::set_read_timer(1);
    } catch {
        #print $logfile "Error thrown: $_ Will retry in 10 seconds.";
        # Adjust this timer for how often to try to reconnect.
        Log::Nxlog::set_read_timer(10);
    };
}
```

Chapter 62. Nessus Vulnerability Scanner

The results of a Nessus scan, saved as XML, can be collected and parsed with NXLog Enterprise Edition.

Scan Sample

```
<?xml version="1.0" ?>
<NessusClientData_v2>
  <Report xmlns:cm="http://www.nessus.org/cm" name="Scan Testbed">
    <ReportHost name="192.168.1.112">
      <HostProperties>
        <tag name="HOST_END">Wed Jun 18 04:20:45 2014</tag>
        <tag name="patch-summary-total-cves">1</tag>
        <tag name="traceroute-hop-1">?</tag>
        <tag name="traceroute-hop-0">10.10.10.20</tag>
        <tag name="operating-system">Linux Kernel</tag>
        <tag name="host-ip">192.168.1.112</tag>
        <tag name="HOST_START">Wed Jun 18 04:19:21 2014</tag>
      </HostProperties>
      <ReportItem port="6667" svc_name="irc" protocol="tcp" severity="0" pluginID="22964"
                   pluginName="Service Detection" pluginFamily="Service detection">
        <description>It was possible to identify the remote service by its banner or by
looking at the error
          message it sends when it receives an HTTP request.
        </description>
        <fname>find_service.nasl</fname>
        <plugin_modification_date>2014/06/03</plugin_modification_date>
        <plugin_name>Service Detection</plugin_name>
        <plugin_publication_date>2007/08/19</plugin_publication_date>
        <plugin_type>remote</plugin_type>
        <risk_factor>None</risk_factor>
        <script_version>$Revision: 1.137 $</script_version>
        <solution>n/a</solution>
        <synopsis>The remote service could be identified.</synopsis>
        <plugin_output>An IRC server seems to be running on this port is running on this
port.</plugin_output>
      </ReportItem>
    </ReportHost>
  </Report>
</NessusClientData_v2>
```

NOTE

While the above sample illustrates the correct syntax, it is not a complete Nessus report. For more information refer to the [Nessus v2 File Format](#) document on [tenable.com](#).

Two examples are shown below for parsing the scans, and both require NXLog Enterprise Edition. The preferred approach uses the [im_perl](#) input module to process the XML; this provides fine-grained control over the collected information. If Perl is not available, the [xm_multiline](#) and [xm_xml](#) extension modules can be used instead.

Example 258. Collecting Events using Perl

This example depicts the preferred way to collect results from a Nessus XML scan file. The [im_perl](#) input module executes a Perl script which reads the Nessus scan. The reports are converted to JSON using the [Exec](#) directive and directed to a file.

This Perl script generates an event for each [ReportItem](#), and includes details from [Report](#) and [ReportHost](#) in each event. Furthermore, normalized [\\$EventTime](#), [\\$Severity](#), and [\\$SeverityValue](#) fields are added to the event record.

nxlog.conf

```

1 <Extension _json>
2   Module      xm_json
3 </Extension>
4
5 <Input perl>
6   Module      im_perl
7   PerlCode   /opt/nxlog/bin/perl-input.pl
8 </Input>
9
10 <Output out>
11  Module     om_file
12  File       "/opt/nxlog/tmp/nessus_report.json"
13  Exec       to_json();
14 </Output>
```

perl-input.pl

```

#!/usr/bin/perl

use strict;
use warnings;

use FindBin;
use lib $FindBin::Bin;
use Log::Nxlog;
use XML::LibXML;

sub read_data {
    my $doc = XML::LibXML->load_xml( location => 'scan.nessus' );
    my $report = $doc->findnodes('/NessusClientData_v2/Report');
    my @nessus_sev = ("INFO", "LOW", "MEDIUM", "HIGH", "CRITICAL");
    my @nxlog_sev_val = (2,3,4,5,5);
    my @nxlog_sev = ("INFO", "WARNING", "ERROR", "CRITICAL", "CRITICAL");
    my %mon2num = qw(
        Jan 01 Feb 02 Mar 03 Apr 04 May 05 Jun 06
        Jul 07 Aug 08 Sep 09 Oct 10 Nov 11 Dec 12
    );
    my $eventtime;

    foreach my $reportHost ( $doc->findnodes('/NessusClientData_v2/Report/ReportHost') )
    {
        $eventtime = "";
        foreach my $properties ( $reportHost->findnodes('./HostProperties/tag') ) {
            if ($properties->getAttribute('name') eq "HOST_END") {
                my($dow, $m, $d, $t, $y) = $properties->textContent =~
                    m<^([a-zA-Z]+) ([a-zA-Z]+) (\d+) (\d+:\d+:\d+) (\d+)$> or die "Not a valid
date";
                $eventtime= $y . "-" . $mon2num{$m} . "-" . $d . " " . $t;
            }
        }
        foreach my $reportItem ( $reportHost->findnodes('./ReportItem') ) {
            my $event = Log::Nxlog::logdata_new();
            my $raw_event = $reportItem->toString();
            Log::Nxlog::set_field_string( $event, 'raw_event', $raw_event );
            Log::Nxlog::set_field_string( $event, 'EventTime', $eventtime );
            Log::Nxlog::set_field_string( $event, 'Report', $report->get_node(1)->getAttribute
('name') );
            Log::Nxlog::set_field_string( $event, 'ReportHost', $reportHost->getAttribute
('name') );
            my @atts = $reportItem->getAttributes();
            
```

```

foreach my $at (@atts) {
    my $na = $at->getName();
    my $va = "" . $at->getValue();
    if ($na eq "severity") {
        my $severity = $va + 0;
        Log::Nxlog::set_field_integer( $event, "NessusSeverityValue", $severity );
        Log::Nxlog::set_field_string( $event, "NessusSeverity", $nessus_sev
[$severity] );
        Log::Nxlog::set_field_integer( $event, "SeverityValue", $nxlog_sev_val
[$severity] );
        Log::Nxlog::set_field_string( $event, "Severity", $nxlog_sev[$severity] );
    } else {
        Log::Nxlog::set_field_string( $event, $na, $va );
    }
}
if ( $reportItem->hasChildNodes ) {
    my @kids = grep { $_->nodeType == XML_ELEMENT_NODE }
    $reportItem->childNodes();
    foreach my $kid (@kids) {
        my $na = $kid->getName();
        my $va = "" . $reportItem->getChildrenByTagName($na);
        Log::Nxlog::set_field_string( $event, $na, $va );
    }
}
Log::Nxlog::add_input_data($event);
}
}
}

```

Output Sample

```
{
  "EventTime": "2014-06-18 04:20:45",
  "Report": "Scan Testbed",
  "ReportHost": "192.168.1.112",
  "port": "6667",
  "svc_name": "irc",
  "protocol": "tcp",
  "NessusSeverityValue": 0,
  "NessusSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "pluginID": "22964",
  "pluginName": "Service Detection",
  "pluginFamily": "Service detection",
  "description": "It was possible to identify the remote service by its banner or by looking at the error\\nmessage it sends when it receives an HTTP request.\n",
  "fname": "find_service.nasl",
  "plugin_modification_date": "2014/06/03",
  "plugin_name": "Service Detection",
  "plugin_publication_date": "2007/08/19",
  "plugin_type": "remote",
  "risk_factor": "None",
  "script_version": "$Revision: 1.137 $",
  "solution": "n/a",
  "synopsis": "The remote service could be identified.",
  "plugin_output": "An IRC server seems to be running on this port is running on this port.",
  "EventReceivedTime": "2017-11-29 20:29:40",
  "SourceModuleName": "perl",
  "SourceModuleType": "im_perl"
}
```

Example 259. Collecting Events using Multiline

This example depicts an alternative way to collect results from Nessus XML scan files, recommended only if Perl is not available. This configuration generates an event for each **ReportItem** found in the scan report.

nxlog.conf

```
1 <Extension multiline_parser>
2     Module      xm_multiline
3     HeaderLine  /^<ReportItem/
4     EndLine     /^</ReportItem>/
5 </Extension>
6
7 <Extension _xml>
8     Module      xm_xml
9     ParseAttributes TRUE
10 </Extension>
11
12 <Extension _json>
13     Module      xm_json
14 </Extension>
15
16 <Input in>
17     Module      im_file
18     File        "nessus_report.xml"
19     InputType   multiline_parser
20     <Exec>
21         # Discard everything that doesn't seem to be an xml event
22         if $raw_event !~ /^<ReportItem/ drop();
23
24         # Parse the xml event
25         parse_xml();
26
27         # Convert to JSON
28         to_json();
29     </Exec>
30 </Input>
31
32 <Output out>
33     Module      om_file
34     File        "nessus_report.json"
35 </Output>
```

Output Sample

```
{  
    "EventReceivedTime": "2017-11-09 10:22:58",  
    "SourceModuleName": "in",  
    "SourceModuleType": "im_file",  
    "ReportItem.port": "6667",  
    "ReportItem.svc_name": "irc",  
    "ReportItem.protocol": "tcp",  
    "ReportItem.severity": "0",  
    "ReportItem.pluginID": "22964",  
    "ReportItem.pluginName": "Service Detection",  
    "ReportItem.pluginFamily": "Service detection",  
    "ReportItem.description": "It was possible to identify the remote service by its banner or by  
looking at the error\nmessage it sends when it receives an HTTP request.\n",  
    "ReportItem.fname": "find_service.nasl",  
    "ReportItem.plugin_modification_date": "2014/06/03",  
    "ReportItem.plugin_name": "Service Detection",  
    "ReportItem.plugin_publication_date": "2007/08/19",  
    "ReportItem.plugin_type": "remote",  
    "ReportItem.risk_factor": "None",  
    "ReportItem.script_version": "$Revision: 1.137 $",  
    "ReportItem.solution": "n/a",  
    "ReportItem.synopsis": "The remote service could be identified.",  
    "ReportItem.plugin_output": "An IRC server seems to be running on this port is running on  
this port."  
}
```

Chapter 63. NetApp

NetApp storage is capable of sending logs to a remote Syslog destination via UDP as well as saving audit logs directly to a network share.

Log Sample

```
4/14/2017 15:40:25 p-netapp1      DEBUG      repl.engine.error: replStatus="8",
replFailureMsg="5898503", replFailureMsgDetail="0", functionName="repl_util::Result
repl_core::Instance::endTransfer(spinnp_uuid_t*)", lineNumber="738"✉
```

For more details about configuring logging on NetApp storage, please refer to the [Product Documentation](#) section of the NetApp Support site. Search for your ONTAP version, which can be determined by running **version -b** from the command line.

Example 260. Checking the ONTAP Version

This example shows the output from ONTAP 8.3.

```
> version -b
/cfcard/x86_64/freebsd/image1/kernel: OS 8.3.1P2
```

63.1. Sending Logs in Syslog Format

The NetApp web interface does not provide a way to configure an external Syslog server, but it is possible to configure this on the command line. This is a cluster level change that only needs to be performed only once per cluster, and will automatically be applied to all members.

NOTE

The steps below have been tested with ONTAP 8 and should work for earlier versions. Exact commands for newer versions may vary.

1. Configure NXLog to receive log entries via UDP and process them as Syslog (see the [examples](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from each member of the cluster.
3. Log in to the cluster address with SSH.
4. Run the following command to configure the Syslog destination. Replace **NAME** and **IP_ADDRESS** with the required values. The default port for UDP is 514.

```
> event destination create -name NAME -syslog IP_ADDRESS
```

5. Now select the messages to be sent. Use the same **NAME** as in the previous step and set **MSG** to the required value.

```
> event route add-destinations -destinations NAME -messagename MSGS
```

A list of messages can be obtained by running the command with a question mark (?) as the argument.

```
> event route add-destinations -destinations NAME -messagename ?
```

It is also possible to specify a severity level in addition to message types. The severity levels are **EMERGENCY**, **ALERT**, **CRITICAL**, **ERROR**, **WARNING**, **NOTICE**, **INFORMATIONAL**, and **DEBUG**.

```
> event route add-destinations -destinations NAME -messagename MSGS
-severity SEVERITY
```

Example 261. Sending Messages at Informational Level to 192.168.6.143

The following commands send all messages with Informational severity level (including higher severities) to 192.168.6.143 in Syslog format via UDP port 514.

```
> event destination create -name nxlog -syslog 192.168.6.143
> event route add-destinations -destinations nxlog -messagename *
  -severity <=INFORMATIONAL
```

Example 262. Receiving Syslog Logs From NetApp

This example shows NetApp Syslog logs as received and processed by NXLog.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10  Module im_udp
11  Host 0.0.0.0
12  Port 514
13  Exec parse_syslog();
14 </Input>
15
16 <Output file>
17  Module om_file
18  File  "/var/log/netapp.log"
19  Exec  to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.61",
  "EventReceivedTime": "2017-04-14 15:38:58",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 0,
  "SyslogFacility": "KERN",
  "SyslogSeverityValue": 7,
  "SyslogSeverity": "DEBUG",
  "SeverityValue": 1,
  "Severity": "DEBUG",
  "Hostname": "192.168.5.61",
  "EventTime": "2017-04-14 15:40:25",
  "Message": "[p-netapp:repl.engine.error:debug]: replStatus=\"8\", replFailureMsg=\"5898503
\", replFailureMsgDetail=\"0\", functionName=\"repl_util::Result
repl_core::Instance::endTransfer(spinnp_uuid_t*)\", lineNumber=\"738\""
}
```

Example 263. Extracting Additional Fields From the Syslog Messages

Messages that contain key-value pairs, like the example at the beginning of the section, can be parsed with

the [xm_kvp](#) module to extract more fields if required.

nxlog.conf

```
1 <Output out>
2   Module      om_null
3 </Output>
4
5 <Extension syslog>
6   Module      xm_syslog
7 </Extension>
8
9 <Extension kvp>
10  Module     xm_kvp
11  KVPDelimiter ,
12  KVDelimiter =
13  EscapeChar \\\
14 </Extension>
15
16 <Input in_syslog_udp>
17  Module     im_udp
18  Host       0.0.0.0
19  Port       514
20  <Exec>
21    parse_syslog();
22    if $Message =~ /(?x)^\[([a-zA-Z0-9-]*)\:(([a-zA-Z.]*)([a-zA-Z]*))\]:
23      \ ([a-zA-Z]+.=.)/
24    {
25      $NAUnit = $1;
26      $NAMsgName = $2;
27      $NAMsgSev = $3;
28      $NAMessage = $4;
29      kvp->parse_kvp($4);
30    }
31  </Exec>
32 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.63",
  "EventReceivedTime": "2017-04-15 23:13:45",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 0,
  "SyslogFacility": "KERN",
  "SyslogSeverityValue": 7,
  "SyslogSeverity": "DEBUG",
  "SeverityValue": 1,
  "Severity": "DEBUG",
  "Hostname": "192.168.5.63",
  "EventTime": "2017-04-15 23:15:14",
  "Message": "[p-netapp3:repl.engine.error:debug]: replStatus=\"5\", replFailureMsg=\"5898500
\", replFailureMsgDetail=\"0\", functionName=\"void
repl_volume::Query::_queryResponse(repl_spinpp::Request&, const spinnp_repl_result_t&,
repl_spinpp::Response*)\", lineNumber=\"149\"",
  "NAUnit": "p-netapp3",
  "NAMsgName": "repl.engine.error",
  "NAMsgSev": "debug",
  "NAMessage": "replStatus=\"5\", replFailureMsg=\"5898500\", replFailureMsgDetail=\"0\",
functionName=\"void repl_volume::Query::_queryResponse(repl_spinpp::Request&, const
spinnp_repl_result_t&, repl_spinpp::Response*)\", lineNumber=\"149\"",
  "replStatus": "5",
  "replFailureMsg": "5898500",
  "replFailureMsgDetail": "0",
  "functionName": "void repl_volume::Query::_queryResponse(repl_spinpp::Request&, const
spinnp_repl_result_t&, repl_spinpp::Response*)",
  "lineNumber": "149"
}
```

63.2. Sending Logs to a Remote File Share

NetApp saves its logs in the Windows EventLog (EVTX) format. In the case of a standalone unit, these logs are available over the network in the `\etc$` share, and can be parsed by the `im_msvisalog` module. However in cluster mode, starting from ONTAP 7, this share is not accessible. Instead, audit logs from each virtual server can be sent to a CIFS share where NXLog can access and read them. This configuration must be performed for each virtual server separately.

To accomplish this, create and enable an audit policy for each virtual server.

```
> vserver audit create -vserver <VIRTUAL_SERVER> -destination <SHARE>
  -rotate-size <SIZE> -rotate-limit <NUMBER>
> vserver audit enable -vserver <VIRTUAL_SERVER>
```

Example 264. Sending NetApp Logs to a CIFS Share

These commands set up an audit policy that sends logs to the specified share, rotates log files at 100 MB, and retains the last 10 rotated log files.

```
> vserver audit create -vserver vs_p12_cifs
  -destination /p-GRT -rotate-size 100M -rotate-limit 10
> vserver audit enable vs_p12_cifs
```

Example 265. Reading Logs From a NetApp EventLog File

This example shows NetApp events as collected and processed by NXLog from an EventLog file.

nxlog.conf

```
1 <Input in_file_evt>
2     Module im_msvistalog
3     File   C:\Temp\NXLog\audit_vs_p12_cifs_last.evtx
4 </Input>
5
6 <Output file_from_eventlog>
7     Module om_file
8     File   "C:\Temp\evt.log"
9     Exec   to_json();
10 </Output>
```

Output Sample

```
{
    "EventTime": "2017-05-10 21:17:12",
    "Hostname": "e3864b4d-8937-11e5-b812-00a098831757/bf4a40a5-9216-11e5-8d9a-00a098831757",
    "Keywords": -9214364837600035000,
    "EventType": "AUDIT_SUCCESS",
    "SeverityValue": 2,
    "Severity": "INFO",
    "EventID": 4624,
    "SourceName": "NetApp-Security-Auditing",
    "ProviderGuid": "{3CB2A168-FE19-4A4E-BDAD-DCF422F13473}",
    "Version": 101,
    "OpcodeValue": 0,
    "RecordNumber": 0,
    "ProcessID": 0,
    "ThreadID": 0,
    "Channel": "Security",
    "ERROR_EVT_UNRESOLVED": true,
    "IpAddress' IPVersion='4": "192.168.17.151",
    "IpPort": "49421",
    "TargetUserSID": "S-1-5-21-4103495029-501085275-2219630704-2697",
    "TargetUserName": "App_Service",
    "TargetUserIsLocal": "false",
    "TargetDomainName": "DOMAIN",
    "AuthenticationPackageName": "KRB5",
    "LogonType": "3",
    "EventReceivedTime": "2017-05-10 22:33:00",
    "SourceModuleName": "in_file_evt",
    "SourceModuleType": "im_msvistalog"
}
```

Chapter 64. .NET Application Logs

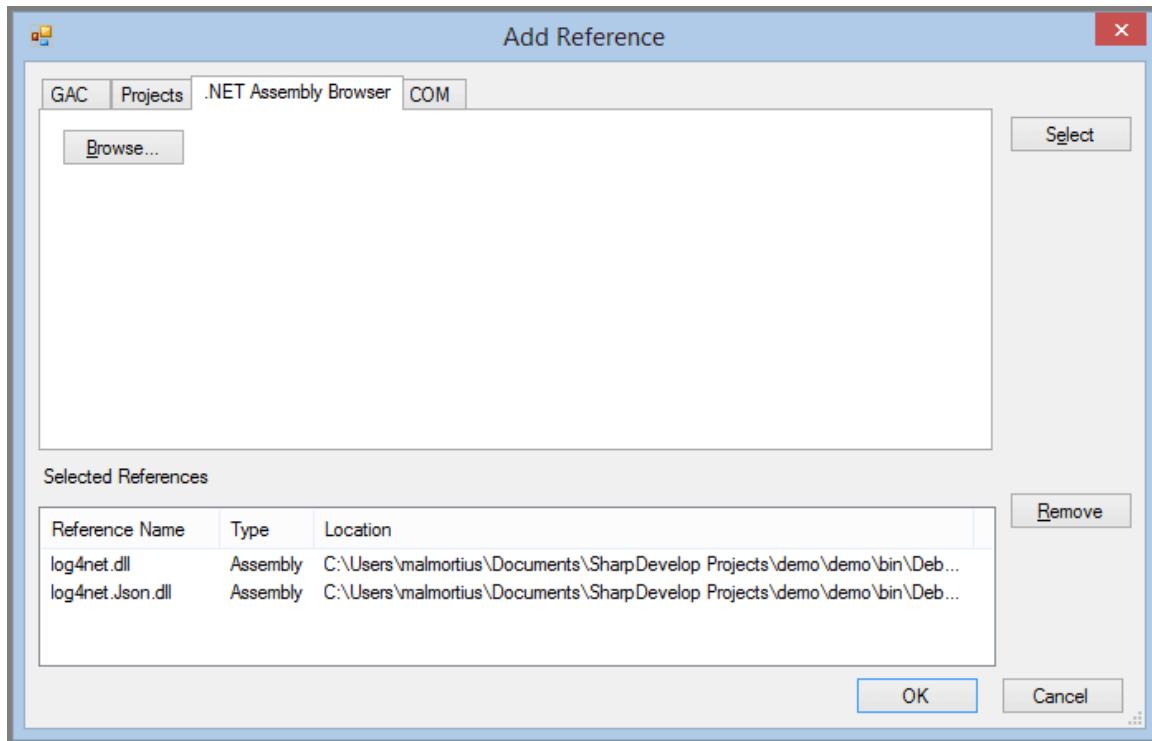
NXLog can be used to capture logs directly from Microsoft .NET™ applications using third-party utilities. This guide demonstrates how to set up these utilities with a sample .NET application with a corresponding NXLog configuration.

This guide uses the [SharpDevelop](#) IDE, but Microsoft Visual Studio™ on Windows, or MonoDevelop on Linux could also be used. The [log4net](#) package and [log4net.Ext.Json](#) extension are also required.

NOTE

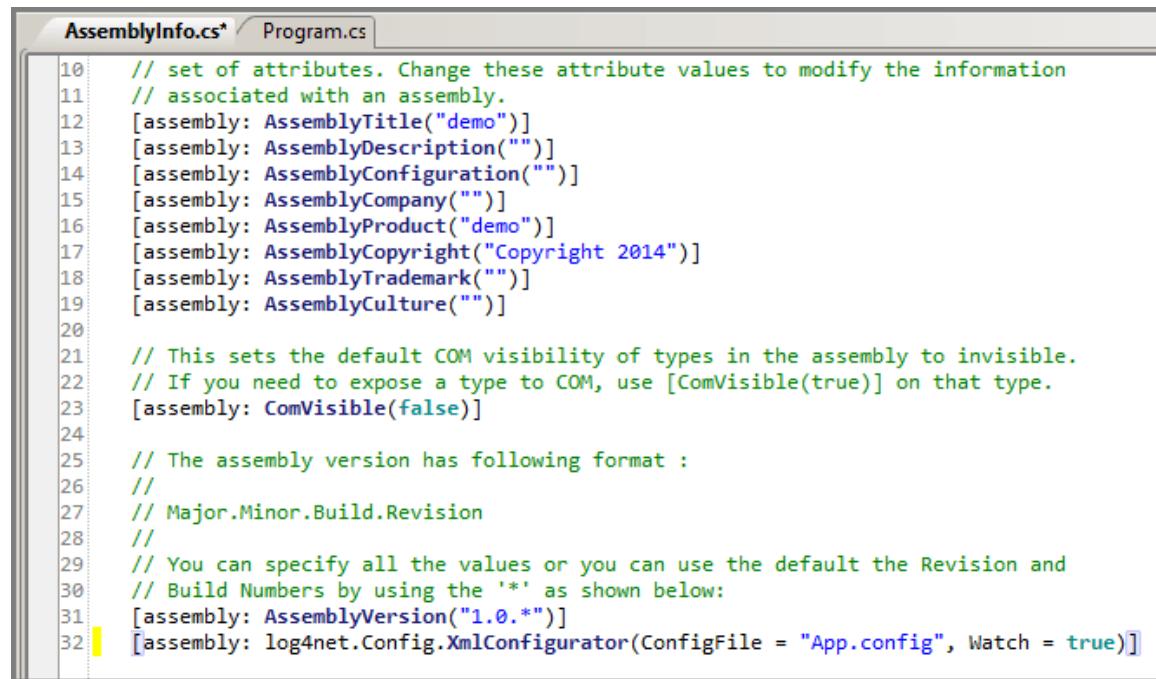
The following instructions were tested with SharpDevelop 5.1.0, .NET 4.5, [log4net 2.0.5](#), and [log4net.Ext.Json 1.2.15.14586](#). To use NuGet packages without the NuGet package manager, simply download the nupkg file using the "Download" link, add a [.zip](#) extension to the file name, and extract.

1. Create a new **Solution** in SharpDevelop by selecting **File > New > Solution** and choosing the **Console Application** option. Enter a **name** and click [**Create**].
2. Place the log4net and log4net.Ext.Json DLL files in the **bin\Debug** directory of your project.
3. Select **Project > Add Reference**. Open the **.NET Assembly Browser** tab and click [**Browse**]. Add the two DLL files so that they appear in the **Selected References** list, then click [**OK**].



4. Edit the **AssemblyInfo.cs** file (under **Properties** in the **Projects** sidebar) and add the following line.

```
[assembly: log4net.Config.XmlConfigurator(ConfigFile = "App.config", Watch = true)]
```



```

AssemblyInfo.cs*  Program.cs
10 // set of attributes. Change these attribute values to modify the information
11 // associated with an assembly.
12 [assembly: AssemblyTitle("demo")]
13 [assembly: AssemblyDescription("")]
14 [assembly: AssemblyConfiguration("")]
15 [assembly: AssemblyCompany("")]
16 [assembly: AssemblyProduct("demo")]
17 [assembly: AssemblyCopyright("Copyright 2014")]
18 [assembly: AssemblyTrademark("")]
19 [assembly: AssemblyCulture("")]
20
21 // This sets the default COM visibility of types in the assembly to invisible.
22 // If you need to expose a type to COM, use [ComVisible(true)] on that type.
23 [assembly: ComVisible(false)]
24
25 // The assembly version has following format :
26 //
27 // Major.Minor.Build.Revision
28 //
29 // You can specify all the values or you can use the default the Revision and
30 // Build Numbers by using the '*' as shown below:
31 [assembly: AssemblyVersion("1.0.*")]
32 [assembly: log4net.Config.XmlConfigurator(ConfigFile = "App.config", Watch = true)]

```

5. Click the Refresh icon in the **Projects** sidebar to show all project files.
6. Create a file named **App.config** in the **bin\Debug** folder, open it for editing, and add the following code. Update the **remoteAddress** value of the with the IP address (or host name) of the NXLog instance.

App.config

```

<configuration>
  <configSections>
    <section name="log4net"
             type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
  </configSections>

  <log4net>
    <appender name="UdpAppender" type="log4net.Appender.UdpAppender">
      <remoteAddress value="192.168.56.103" />
      <remotePort value="514" />
      <layout type="log4net.Layout.SerializedLayout, log4net.Ext.Json" />
    </appender>

    <root>
      <level value="DEBUG" />
      <appender-ref ref="UdpAppender" />
    </root>
  </log4net>
</configuration>

```

7. Edit the **Program.cs** file, and replace its contents with the following code. This loads the log4net module and creates some sample log messages.

Program.cs

```

using System;
using log4net;

namespace demo
{
    class Program
    {
        private static readonly log4net.ILog mylog = log4net.LogManager.GetLogger(typeof(Program));
        public static void Main(string[] args)
        {
            log4net.Config.BasicConfigurator.Configure();
            mylog.Debug("This is a debug message");
            mylog.Warn("This is a warn message");
            mylog.Error("This is an error message");
            mylog.Fatal("This is a fatal message");
            Console.ReadLine();
        }
    }
}

```

8. Configure NXLog.

nxlog.conf

```

1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Input in>
6     Module im_udp
7     Host   0.0.0.0
8     Port   514
9     <Exec>
10        $raw_event =~ s/\s+$/;;
11
12        # Parse JSON into fields for later processing if required
13        parse_json();
14    </Exec>
15 </Input>
16
17 <Output out>
18     Module om_file
19     File   "/tmp/output"
20 </Output>
21
22 <Route r>
23     Path   in => out
24 </Route>

```

9. In SharpDevelop, press the F5 key to build and run the application. The following output should appear.

Demo Output

```

4301 [1] DEBUG demo.Program (null) - This is a debug message<br/>
4424 [1] WARN demo.Program (null) - This is a warn message<br/>
4425 [1] ERROR demo.Program (null) - This is an error message<br/>
4426 [1] FATAL demo.Program (null) - This is a fatal message<br/>

```

10. Examine the /tmp/output file. It should show the sample log entries produced by the .NET application.

NXLog Output

```
{"date":"2014-03-19T09:41:08.7231787+01:00","Level":"DEBUG","AppDomain":"demo.exe","Logger":"demo.Program","ThreadID":1,"Message":"This is a debug message","Exception":""}←  
{"date":"2014-03-19T09:41:08.8456254+01:00","Level":"WARN","AppDomain":"demo.exe","Logger":"demo.Program","ThreadId":1,"Message":"This is a warn message","Exception":""}←  
{"date":"2014-03-19T09:41:08.8466327+01:00","Level":"ERROR","AppDomain":"demo.exe","Logger":"demo.Program","ThreadId":1,"Message":"This is an error message","Exception":""}←  
{"date":"2014-03-19T09:41:08.8476223+01:00","Level":"FATAL","AppDomain":"demo.exe","Logger":"demo.Program","ThreadId":1,"Message":"This is a fatal message","Exception":""}←
```

Chapter 65. Nginx

The Nginx web server supports error and access logging. Both types of logs can be written to file, or forwarded as Syslog via UDP, or written as Syslog to a Unix domain socket. The sections below provide a brief overview; see the [Logging](#) section of the Nginx documentation for more detailed information.

65.1. Error Log

The `error_log` directive configures the destination and log level for the error log. This directive can be given in the main (top-level) configuration context to override the default. It can also be specified at the `http`, `stream`, `server`, and `location` levels, where it will override the inherited setting from the higher levels.

Example 266. Collecting Error Logs From File

With the following directive, Nginx will log all messages of "warn" severity or higher to the specified log file.

`nginx.conf`

```
error_log /var/log/nginx/error.log warn;
```

Following is a log message generated by Nginx, an NXLog configuration for parsing it, and the resulting JSON.

Log Sample

```
2017/08/07 04:37:16 [emerg] 17479#17479: epoll_create() failed (24: Too many open files)↔
```

`nxlog.conf`

```
<Input nginx_error>
    Module  im_file
    File    '/var/log/nginx/error.log'
    <Exec>
        if $raw_event =~ /^(\S+ \S+) \[(\S+)\] (\d+)\#(\d+): (\*(\d+ )?(.+)$/
        {
            $EventTime = strftime($1, '%Y/%m/%d %H:%M:%S');
            $NginxLogLevel = $2;
            $NginxPID = $3;
            $NginxTID = $4;
            if $6 != '' $NginxCID = $6;
            $Message = $7;
        }
    </Exec>
</Input>
```

Output Sample

```
{
    "EventReceivedTime": "2017-08-07T04:37:16.245375+02:00",
    "SourceModuleName": "nginx_error",
    "SourceModuleType": "im_file",
    "EventTime": "2017-08-07T04:37:16.000000+02:00",
    "NginxLogLevel": "emerg",
    "NginxPID": "17479",
    "NginxTID": "17479",
    "Message": "epoll_create() failed (24: Too many open files)"}
```

Example 267. Collecting Error Logs via Syslog

With this directive, Nginx will forward all messages of "warn" severity or higher to the specified Syslog server. The messages will be generated with the "local7" facility.

nginx.conf

```
error_log syslog:server=192.168.1.1:514,facility=local7 warn;
```

This NXLog configuration can be used to parse the logs.

nxlog.conf

```
<Input nginx_error>
    Module im_udp
    Host 0.0.0.0
    Port 514
    <Exec>
        parse_syslog();
        if $Message =~ /^$S+ $S+ [\S+] (\d+)\#(\d+): (\*(\d+) )?(.+)$/
        {
            $NginxPID = $1;
            $NginxTID = $2;
            if $4 != '' $NginxCID = $4;
            $Message = $5;
        }
    </Exec>
</Input>
```

Output Sample

```
{
    "MessageSourceAddress": "192.168.1.12",
    "EventReceivedTime": "2017-08-07T04:37:16.441368+02:00",
    "SourceModuleName": "nginx_error",
    "SourceModuleType": "im_udp",
    "SyslogFacilityValue": 23,
    "SyslogFacility": "LOCAL7",
    "SyslogSeverityValue": 1,
    "SyslogSeverity": "ALERT",
    "SeverityValue": 5,
    "Severity": "CRITICAL",
    "Hostname": "nginx-host",
    "EventTime": "2017-08-07T04:37:16.000000+02:00",
    "SourceName": "nginx",
    "Message": "epoll_create() failed (24: Too many open files)",
    "NginxPID": "17479",
    "NginxTID": "17479"
}
```

Example 268. Collecting Error Logs via Unix Domain Socket

With this directive, Nginx will forward all messages of "warn" severity or higher to the specified Unix domain socket. The messages will be sent in Syslog format with the "local7" Syslog facility.

nginx.conf

```
error_log syslog:server=unix:/var/log/nginx/error.sock,facility=local7 warn;
```

nxlog.conf

```
<Input nginx_error>
    Module im_uds
    UDS /var/log/nginx/error.sock
    <Exec>
        parse_syslog();
        if $Message =~ /^$S+ $S+ \[$S+\] ($d+)\#($d+): ($*(\d+) )?(.+)$/
        {
            $NginxPID = $1;
            $NginxTID = $2;
            if $4 != '' $NginxCID = $4;
            $Message = $5;
        }
    </Exec>
</Input>
```

65.2. Access Log

By default, Nginx writes access logs to [logs/access.log](#) in the Combined Log Format. An NXLog configuration example for parsing this can be found in the [Common & Combined Log Formats](#) section. Access logs can also be forwarded in Syslog format via UDP or a Unix domain socket, as shown below.

The log format can be customized by setting the [log_format](#) directive; see the [Nginx documentation](#) for more information.

Example 269. Collecting Access Logs via Syslog

With this directive, Nginx will forward access logs to the specified Syslog server. The messages will be generated with the "local7" facility and the "info" severity.

nginx.conf

```
access_log syslog:server=192.168.1.1:514,facility=local7,severity=info;
```

This NXLog configuration can be used to parse the logs.

nxlog.conf

```
<Input nginx_access>
    Module im_udp
    Host 0.0.0.0
    Port 514
    <Exec>
        parse_syslog();
        if $Message =~ /(?x)^(S+)\S+\S+\[(^\]+)\]\"(S+)\(.+)
            \ HTTP/\d.\d\"(\S+)\S+\\"([^\"]+)\"
            \ \"([^\"]+)\"
        {
            $Hostname = $1;
            if $2 != '-' $AccountName = $2;
            $EventTime = parsedate($3);
            $HTTPMethod = $4;
            $HTTPURL = $5;
            $HTTPResponseStatus = $6;
            if $7 != '-' $FileSize = $7;
            if $8 != '-' $HTTPReferer = $8;
            if $9 != '-' $HTTPUserAgent = $9;
            delete($Message);
        }
    </Exec>
</Input>
```

Output Sample

```
{
    "MessageSourceAddress": "192.168.1.12",
    "EventReceivedTime": "2017-08-07T06:15:55.662319+02:00",
    "SourceModuleName": "nginx_access",
    "SourceModuleType": "im_udp",
    "SyslogFacilityValue": 23,
    "SyslogFacility": "LOCAL7",
    "SyslogSeverityValue": 6,
    "SyslogSeverity": "INFO",
    "SeverityValue": 2,
    "Severity": "INFO",
    "Hostname": "192.168.1.12",
    "EventTime": "2017-08-07T06:15:55.000000+02:00",
    "SourceName": "nginx",
    "HTTPMethod": "GET",
    "HTTPURL": "/",
    "HTTPResponseStatus": "304",
    "FileSize": "0",
    "HTTPUserAgent": "Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0"
}
```

Example 270. Collecting Access Logs via Unix Domain Socket

With this directive, Nginx will forward all messages of "warn" severity or higher to the specified Unix domain socket. The messages will be sent in Syslog format with the "local7" Syslog facility.

nginx.conf

```
access_log syslog:server=unix:/var/log/nginx/access.sock,facility=local7,severity=info;
```

nxlog.conf

```
<Input nginx_access>
    Module im_uds
    UDS /var/log/nginx/access.sock
    <Exec>
        parse_syslog();
        if $Message =~ /(?x)^(\$+)\ \$+\ (\$+)\ \$+([([^\ ]]+))\$ \("$(\$+)\ \$\ .+)
            \ HTTP\ /\d\.\d\" \("$+\ \$+\ \"$([^\ ]+)\"
            \ \"$([^\ ]+)\"
        {
            $Hostname = $1;
            if $2 != '-' $AccountName = $2;
            $EventTime = parsedate($3);
            $HTTPMethod = $4;
            $HTTPURL = $5;
            $HTTPResponseStatus = $6;
            if $7 != '-' $FileSize = $7;
            if $8 != '-' $HTTPReferer = $8;
            if $9 != '-' $HTTPUserAgent = $9;
            delete($Message);
        }
    </Exec>
</Input>
```

Chapter 66. Postfix

NXLog can be configured to collect logs from the Postfix mail server. Postfix logs its actions to the standard system logger with the **mail** facility type.

Syslog/Postfix Log Format

```
Oct 10 01:23:45 hostname postfix/component[pid]: message↔
```

The **component** indicates the Postfix process that produced the log message. Most log entries, those relevant to particular email messages, also include the queue ID of the email message as the first part of the message.

Log Sample

```
Oct 10 01:23:45 mailhost postfix/smtpd[2534]: 4F9D195432C: client=localhost[127.0.0.1]↔
Oct 10 01:23:45 mailhost postfix/cleanup[2536]: 4F9D195432C: message-
id=<20161001103311.4F9D195432C@mail.example.com>↔
Oct 10 01:23:46 mailhost postfix/qmgr[2531]: 4F9D195432C: from=<origin@other.com>, size=344, nrcpt=1
(queue active)↔
Oct 10 01:23:46 mailhost postfix/smtp[2538]: 4F9D195432C: to=<destination@example.com>,
relay=mail.example.com[216.150.150.131], delay=11, status=sent (250 Ok: queued as 8BDCA22DA71)↔
```

66.1. Configuring Postfix Logging

Several configuration directives, set in **main.cf**, can be used to adjust Postfix's logging behavior.

lsmtp_tls_loglevel

smtp_tls_loglevel

smtpd_tls_loglevel

The **loglevel** directives should be set to **0** (disabled, the default) or **1** during normal operation. Values of **2** or **3** can be used for troubleshooting.

debug_peer_level

Specify the increment in logging level when a remote client or server matches a pattern in the **debug_peer_list** parameter (default **2**).

debug_peer_list

Provide a list of remote client or server hostnames or network address patterns for which to increase the logging level.

See the [Postfix Debugging Howto](#) and the [postconf\(5\) man page](#) for more information.

66.2. Collecting and Processing Postfix Logs

The local syslogd configuration determines where and how the **mail** facility logs are written, but normally the logs can be found in **/var/log/maillog** or **/var/log/mail.log**. See [Collecting and Parsing Syslog](#) and [Linux System Logs](#) for more information about collecting Syslog logs.

Example 271. Reading From Syslog Log File

This configuration reads the Postfix logs from file and forwards them via TCP to a remote host.

nxlog.conf

```

1 <Input postfix>
2   Module im_file
3   File   "/var/log/mail.log"
4 </Input>
5
6 <Output out>
7   Module om_tcp
8   Host   192.168.1.1
9   Port   1514
10 </Output>
```

It is also possible to parse individual Postfix messages into fields, providing access to more fine-grained filtering and analysis of log data. The NXLog **Exec** directive can be used to apply regular expressions for this purpose.

Example 272. Extracting Additional Fields and Filtering

Here is the Input module instance again, extended to parse the Postfix messages in the example above. Various fields are added to the event record, depending on the particular message received. Then in the Output module instance, only those log entries that are from Postfix's **smtp** component and are being relayed through **mail.example.com** are logged to the output file.

nxlog.conf

```

1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 <Input postfix>
6   Module im_file
7   File   "/var/log/mail.log"
8   <Exec>
9     if $raw_event =~ /(?x)^(\S+\ +\d+\ \d+:\d+:\d+)\ (\S+)
10      \ postfix\//(\S+)\[(\d+)\]:\ (.+)\$/
11    {
12      $EventTime = parsedate($1);
13      $HostName = $2;
14      $SourceName = "postfix";
15      $Component = $3;
16      $ProcessID = $4;
17      $Message = $5;
18      if $Component == "smtpd" and
19        $Message =~ /(\w+): client=(\S+)\[((\d.)+)\]/
20      {
21        $QueueID = $1;
22        $ClientHostname = $2;
23        $ClientIP = $3;
24      }
25      if $Component == "cleanup" and
26        $Message =~ /(\w+): message-id=(<\S+@\S+>)/
27      {
28        $QueueID = $1;
29        $MessageID = $2;
30      }
31      if $Component == "qmgr" and
```

```

32     $Message =~/( \w+): from=(\S+@\S+>), size=(\d+), nrcpt=(\w+)/
33     {
34         $QueueID = $1;
35         $Sender = $2;
36         $Size = $3;
37         $Nrcpt = $4;
38     }
39     if $Component == "smtp" and
40         $Message =~ /(?x)(\w+):\ \ to=(\S+@\S+>),\ relay=([\w.]+)\[([ \d.]+)\],
41             \ delay=(\d+),\ status=(\w+)\ \((\d+)\)\ \w+:\ queued\ as
42             \ (\w+)\)/
43     {
44         $QueueID = $1;
45         $Recipient = $2;
46         $RelayHostname = $3;
47         $RelayIP = $4;
48         $Delay = $5;
49         $Status = $6;
50         $SMTPCode = $7;
51         $QueueIDDelivered = $8;
52     }
53 }
54 </Exec>
55 </Input>
56
57 <Output out>
58     Module om_file
59     File   "/var/log/smtp.log"
60     <Exec>
61         if $Component != "smtp" drop();
62         if $RelayHostname != "mail.example.com" drop();
63         to_json();
64     </Exec>
65 </Output>
```

Using the example log entries above, this configuration results in a single JSON entry written to the log file.

Output Sample

```
{
    "EventReceivedTime": "2016-10-05 16:38:57",
    "SourceModuleName": "postfix",
    "SourceModuleType": "im_file",
    "EventTime": "2016-10-10 01:23:46",
    "HostName": "mail",
    "SourceName": "postfix",
    "Component": "smtp",
    "ProcessID": "2538",
    "Message": "4F9D195432C: to=<destination@example.com>, relay=mail.example.com[216.150.150.131], delay=11, status=sent (250 Ok: queued as 8BDCA22DA71)",
    "QueueID": "4F9D195432C",
    "Recipient": "<destination@example.com>",
    "RelayHostname": "mail.example.com",
    "RelayIP": "216.150.150.131",
    "Delay": "11",
    "Status": "sent",
    "SMTPCode": "250",
    "QueueIDDelivered": "8BDCA22DA71"
}
```

Chapter 67. Promise

The Promise Storage Area Network (SAN) is capable of sending SNMP traps to remote destinations. Unfortunately Syslog is not supported on these units.

Log Sample

```
2654 Fan 4 Enc 1      Info     Apr 27, 2017 19:08:48 PSU fan or blower speed is decreased
```

There is a single management interface no matter how many shelves are installed, so configuration only needs to be performed once from the Promise web interface or the command line.

More information about configuring Promise arrays is available in the [E-Class product manual](#). Also, additional details on CNMP configuration and links to MIB files are available in the following [KB article](#).

1. Configure NXLog for receiving SNMP traps (see the [example](#) below). Remember to place the MIB file in the directory specified by the [MIBDir](#) directive. Then restart NXLog.
2. Make sure the NXLog agent is accessible from the unit.
3. Configure Promise by using the web interface or the command line. See the following sections.

Example 273. Receiving SNMP Traps From Promise

This example shows SNMP trap messages from Promise, as received and processed by NXLog.

```
nxlog.conf
1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module      xm_json
7 </Extension>
8
9 <Extension snmp>
10  Module      xm_snmp
11  MIBDir     /usr/share/mibs/iana
12 </Extension>
13
14 <Input in_snmp_udp>
15   Module      im_udp
16   Host        0.0.0.0
17   Port        162
18   InputType   snmp
19   Exec        parse_syslog();
20 </Input>
21
22 <Output file_snmp>
23   Module      om_file
24   File        "/var/log/snmp.log"
25   Exec        to_json();
26 </Output>
```

Output Sample

```
{  
    "SNMP.CommunityString": "public",  
    "SNMP.RequestID": 1295816642,  
    "EventTime": "2017-04-27 20:44:37",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "OID.1.3.6.1.2.1.1.3.0": 67,  
    "OID.1.3.6.1.6.3.1.1.4.1.0": "1.3.6.1.4.1.7933.1.20.0.11.0.1",  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.1": 2654,  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.2": 327683,  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.3": 327683,  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.4": 2,  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.5": "Fan 4 Enc 1",  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.6": "Apr 27, 2017 19:08:48",  
    "OID.1.3.6.1.4.1.7933.1.20.0.10.7": "PSU fan or blower speed is decreased",  
    "MessageSourceAddress": "192.168.10.21",  
    "EventReceivedTime": "2017-04-27 20:44:37",  
    "SourceModuleName": "in_snmp_udp",  
    "SourceModuleType": "im_udp",  
    "SyslogFacilityValue": 1,  
    "SyslogFacility": "USER",  
    "SyslogSeverityValue": 5,  
    "SyslogSeverity": "NOTICE",  
    "Hostname": "INFO",  
    "Message": "OID.1.3.6.1.2.1.1.3.0=\\"67\\" OID.1.3.6.1.6.3.1.1.4.1.0=  
\\"1.3.6.1.4.1.7933.1.20.0.11.0.1\\" OID.1.3.6.1.4.1.7933.1.20.0.10.1=\\"2654\\"  
OID.1.3.6.1.4.1.7933.1.20.0.10.2=\\"327683\\" OID.1.3.6.1.4.1.7933.1.20.0.10.3=\\"327683\\"  
OID.1.3.6.1.4.1.7933.1.20.0.10.4=\\"2\\" OID.1.3.6.1.4.1.7933.1.20.0.10.5=\\"Fan 4 Enc 1\\"  
OID.1.3.6.1.4.1.7933.1.20.0.10.6=\\"Apr 27, 2017 19:08:48\\" OID.1.3.6.1.4.1.7933.1.20.0.10.7=  
\\"PSU fan or blower speed is decreased\\""  
}
```

NOTE

The steps below have been tested on the VTrak E600 series and should work on other models as well.

67.1. Configuring via Web Interface

Follow these steps to enable sending SNMP traps through the web interface.

1. Log in to the web interface.
2. Go to **Subsystems > Administrative Tools > Software Management**.
3. Under the **Service** tab, click on **[SNMP]**.
4. Under **Trap Sink**, specify the **Trap Sink Server** IP address and select the appropriate **Trap Filter** to choose the logging level. Then click **[Update]**.

SNMP Settings

Service **Export** **Import** **Firmware Update** **Image Version** **Help**

Service Status -- SNMP

Running Status: Started **Stop** **Restart**

Service Setting -- SNMP

Startup Type: Automatic Manual

SNMP Overall Settings

Port	2
System Name	Email
System Location	Q6
System Contact	admin
Read Community	public
Write Community	.

Trap Sink

Trap Sink Server: 192.168.6.143

Trap Filter: Info Warning Minor Major Critical Fatal

Trap Sinks

#	Trap Sink Server	Trap Filter
No trap sink available.		

Buttons: **Reset** **Submit** **Cancel**

5. Make sure **Running Status** is **Started** and **Startup Type** is set to **Automatic**.
6. Click [**Submit**] and confirm SNMP restart.

67.2. Configuring via Command Line

Follow these steps to enable sending SNMP traps through the command line interface.

1. Connect to Promise via SSH.
2. Type **menu**.
3. Go to **Additional Info and Management > Software Management > SNMP**.
4. Select **Trap Sinks > Create New Trap Sink**.
5. Specify the remote IP address under **Trap Sink Server** and the logging level under **Trap Filter**.
6. Select **Save SNMP Trap Sink**.
7. Select **Return to Previous Menu** and then **Restart**.
8. Make sure **Startup Type** is set to **Automatic**.

Chapter 68. SafeNet KeySecure

SafeNet KeySecure devices are capable of sending their logs to a remote Syslog destination via UDP or TCP. KeySecure has four different logs: System, Audit, Activity, and Client Event. Each one has a slightly different format, and each can be configured with up to two Syslog servers. There is also an option to sign and encrypt logs messages before sending them to the remote destination. Configuration for this type of scenario is outside of the scope of this section.

Sample Audit Message

```
2017-03-26 18:12:04 [admin] [Login] [CLI]: Logged out from 192.168.15.231 via SSH↔
```

In case of a cluster with two or more KeySecure devices, the configuration change on one of them will be replicated to other members. Each member will be sending logs separately. For more details regarding logging configuration on SafeNet KeySecure, refer to the [KeySecure Appliance User Guide](#).

NOTE

This section covers configuration for sending logs via UDP. To use TCP instead, just select it instead where appropriate.

1. Configure NXLog for receiving Syslog logs (see the [examples](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from the KeySecure device.
3. Configure Syslog logging on KeySecure using either the web interface or the command line. See the following sections.

NOTE

The steps in the following sections have been tested on KeySecure 460 and should work on other models also.

Example 274. Receiving Logs From KeySecure

This example shows a KeySecure Audit log message as received and processed by NXLog. Use the **im_tcp** module instead of **im_udp** to receive Syslog messages via TCP instead.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10    Module im_udp
11    Host   0.0.0.0
12    Port   514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/keysecure.log"
19    Exec   to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.20",
  "EventReceivedTime": "2017-03-26 18:11:36",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 17,
  "SyslogFacility": "LOCAL1",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "p-keysecure1",
  "EventTime": "2017-03-26 18:12:26",
  "SourceName": "IngrianAudit",
  "Message": "2017-03-26 18:12:26 [admin] [Login] [CLI]: Logged in from 192.168.15.231 via SSH"
}
```

Example 275. Extracting Additional Fields

Additional field extraction can also be configured. Note that this depends on which particular log the message is coming from, as each has a different format.

nxlog.conf

```

1 <Input in_syslog_udp>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Exec>
6     parse_syslog();
7     if $Message =~ /(?x)^$d{4}-$d{2}-$d{2}\ $d{2}:\$d{2}:\$d{2}\ \[(([a-zA-Z])*))\]
8       \[(([a-zA-Z])*))\ \[(([a-zA-Z])*))\:(.*$)/
9     {
10       $KSUsername = $1;
11       $KSEvent = $2;
12       $KSSubsys = $3;
13       $KSMessage = $4;
14     }
15   </Exec>
16 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.5.20",
  "EventReceivedTime": "2017-04-15 19:14:59",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 17,
  "SyslogFacility": "LOCAL1",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "p-keysecure1",
  "EventTime": "2017-04-15 19:16:29",
  "SourceName": "IngrianAudit",
  "Message": "2017-04-15 19:16:29 [admin] [Login] [CLI]: Logged in from 192.168.15.231 via SSH",
  "KSUsername": "admin",
  "KSEvent": "Login",
  "KSSubsys": "CLI",
  "KSMessage": "Logged in from 192.168.15.231 via SSH"
}
```

68.1. Configuring via the Web Interface

1. Log in to the KeySecure Management Console.
2. Go to **Device > Logs & Statistics > Log Configuration > Rotation & Syslog**.
3. Select a log type and click **[Edit]** to change the settings.
4. Select the **Enable Syslog** option and specify the required IP addresses, ports, protocols, and facility for up to two servers.

Syslog Settings

Log Name	Enable Syslog	Syslog Server #1 IP	Syslog Server #1 Port	Server #1 Proto	Syslog Server #2 IP	Syslog Server #2 Port	Server #2 Proto	Syslog Facility
System	<input checked="" type="checkbox"/>	p-syslog1	514	udp	[None]	514	udp	local1
Audit	<input checked="" type="checkbox"/>	192.168.15.223	514	udp		514	udp	local1
Activity	<input checked="" type="checkbox"/>	p-syslog1	514	udp	[None]	514	udp	local1
Client Event	<input type="checkbox"/>	p-syslog1	514	udp	[None]	514	udp	local1

Save **Cancel**

5. Click **Save**.
6. Repeat for the other log types as required.

68.2. Configuring via the Command Line

1. Log in to KeySecure via SSH.
2. Run the following commands. Follow the prompts to enable remote syslog with the required IP addresses, ports, protocols, and facility for up to two servers.

```
# configure
# system syslog
# audit syslog
# activity syslog
# clientevent syslog
```

Example 276. Forwarding System Logs

The following commands enable sending System logs to 192.168.6.43 via UDP port 514.

```
p-keysecure1# configure
p-keysecure1 (config)# system syslog
Enable Syslog [y]:
Syslog Server #1 IP: 192.168.6.143
Syslog Server #1 Port [514]:
Server #1 Proto:
  1: udp
  2: tcp
Enter a number (1 - 2) [1]:
Syslog Server #2 IP:
Syslog Server #2 Port [514]:
Server #2 Proto:
  1: udp
  2: tcp
Enter a number (1 - 2) [1]:
Syslog Facility:
  1: local0
  2: local1
  3: local2
  4: local3
  5: local4
  6: local5
  7: local6
  8: local7
Enter a number (1 - 8) [2]:
System Log syslog settings successfully saved.  Syslog is enabled.
Warning: The syslog protocol insecurely transfers logs in cleartext
```

Chapter 69. Snare

The Snare Agent is a popular log collection software for Windows EventLog. The Snare format is supported by many tools and SIEM vendors. It uses tab delimited records and can use Syslog as the transport. NXLog can be configured to collect or forward logs in the Snare format.

The Snare format can be used with or without the Syslog header.

Snare Format

```
HOSTNAME → MSWinEventLog → Criticality → EventLogSource → SnareCounter → SubmitTime → EventID →  
SourceName → UserName → SIDType → EventLogType → ComputerName → CategoryString → DataString →  
ExpandedString → OptionalMD5Checksum←
```

"Snare Over Syslog" Format

```
<PRI>TIMESTAMP HOSTNAME MSWinEventLog → Criticality → EventLogSource → SnareCounter → SubmitTime →  
EventID → SourceName → UserName → SIDType → EventLogType → ComputerName → CategoryString →  
DataString → ExpandedString → OptionalMD5Checksum←
```

69.1. Collecting Snare

NXLog can parse Snare logs with the [parse_csv\(\)](#) procedure provided by the *xm_csv* extension module.

Example 277. Using xm_csv to Capture Snare Logs

With the following configuration, NXLog will accept Snare format logs via UDP, parse them, convert to JSON, and output the result to file. This configuration supports both "Snare over Syslog" and the regular Snare format.

nxlog.conf

```

1 <Extension snare>
2   Module      xm_csv
3   Fields      $MSWINEventLog, $Criticality, $EventLogSource, $SnareCounter, \
4               $SubmitTime, $EventID, $SourceName, $UserName, $SIDType, \
5               $EventLogType, $ComputerName, $Category, $Data, $Expanded, \
6               $MD5Checksum
7   FieldTypes  string, integer, string, integer, datetime, integer, string, \
8               string, string, string, string, string, string, string, string
9   Delimiter   \t
10 </Extension>
11
12 <Extension json>
13   Module      xm_json
14 </Extension>
15
16 <Extension syslog>
17   Module      xm_syslog
18 </Extension>
19
20 <Input in>
21   Module      im_udp
22   Host        0.0.0.0
23   Port        6161
24   <Exec>
25     parse_syslog_bsd();
26     if $Message =~ /^((\w+)\t)?(MSWinEventLog.+)$/
27     {
28       if $2 != ''
29       {
30         $Hostname = $2;
31         $Message = $3;
32       }
33       snare->parse_csv($Message);
34       $Message = $Expanded;
35     }
36   </Exec>
37 </Input>
38
39 <Output out>
40   Module      om_file
41   File        '/var/log/json'
42   Exec        to_json();
43 </Output>
44
45 <Route r>
46   Path        in => out
47 </Route>
```

Input Sample ("Snare Over Syslog")

```
<13>Nov 21 11:40:27 myserver MSWinEventLog => 0 => Security => 32 => Mon Nov 21 11:40:27 2016 =>
592 => Security => Andy => User => Success Audit => MAIN => DetailedTracking => Process ended =>
Ended process ID: 2455<
```

Output Sample

```
{  
    "EventReceivedTime": "2016-11-21 11:40:28",  
    "SourceModuleName": "in",  
    "SourceModuleType": "im_file",  
    "SyslogFacilityValue": 1,  
    "SyslogFacility": "USER",  
    "SyslogSeverityValue": 5,  
    "SyslogSeverity": "NOTICE",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "Hostname": "myserver",  
    "EventTime": "2016-11-21 11:40:27",  
    "Message": "Ended process ID: 2455",  
    "MSWINEventLog": "MSWinEventLog",  
    "Criticality": 0,  
    "EventLogSource": "Security",  
    "SnareCounter": 32,  
    "SubmitTime": "2016-11-21 11:40:27",  
    "EventID": 592,  
    "SourceName": "Security",  
    "UserName": "Andy",  
    "SIDType": "User",  
    "EventLogType": "SuccessAudit",  
    "ComputerName": "MAIN",  
    "CategoryString": "DetailedTracking",  
    "DataString": "Process ended",  
    "ExpandedString": "Ended process ID: 2455"  
}
```

69.2. Generating Snare

NXLog can also generate Snare logs in place of the original Snare agent with the `to_syslog_snare()` procedure provided by the `xm_syslog` extension module.

Example 278. Sending EventLog in Snare Format

With this configuration, NXLog will read the Windows EventLog, convert it to Snare format, and output it via UDP. NXLog log messages are also included (via the [im_internal](#) module). Tabs and newline sequences are replaced with spaces.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input internal>
6     Module im_internal
7 </Input>
8
9 <Input eventlog>
10    Module im_msvisalog
11    Exec   $Message =~ s/(\t|\R)/ /g;
12 </Input>
13
14 <Output out>
15    Module om_udp
16    Host   192.168.1.1
17    Port   514
18    Exec   to_syslog_snare();
19 </Output>
20
21 <Route r>
22     Path   internal, eventlog => out
23 </Route>
```

Output Sample

```
<13>Nov 21 11:40:27 myserver MSWinEventLog => 0 => Security => 32 => Mon Nov 21 11:40:27 2016 =>
592 => Security => N/A => N/A => Success Audit => MAIN => DetailedTracking => Process ended => Ended
process ID: 2455<
```

Chapter 70. Snort

NXLog can be used to capture and process logs from the Snort™ network intrusion prevention system.

Snort writes log entries to the `/var/log/snort/alert` file. Each entry contains the date and time of the event, the packet header, a description of the type of breach that was detected, and a severity rating. Each log entry traverses multiple lines, and there is neither a fixed number of lines nor a separator.

Example 279. Snort Rules and Log Samples

Following are three example Snort rules and corresponding log messages.

Snort Rule

```
alert icmp any any -> any any (msg:"ICMP Packet"; sid:477; rev:3;)
```

Log Sample

```
[**] [1:477:3] ICMP Packet [**]↵
[Priority: 0]↵
04/30-07:54:41.759229 172.25.212.245 -> 172.25.212.153↵
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:96 DF↵
Type:8 Code:0 ID:16348 Seq:0 ECHO↵
```

Snort Rule

```
alert tcp any any -> any any (msg:"Exploit detected"; sid:1000001; content:"exploit";)
```

Log Sample

```
[**] [1:1000001:0] Exploit detected [**]↵
[Priority: 0]↵
04/30-07:54:38.312536 172.25.212.204:80 -> 192.168.255.110:46127↵
TCP TTL:64 TOS:0x0 ID:19844 IpLen:20 DgmLen:505 DF↵
***AP*** Seq: 0xF936BE12 Ack: 0x2C9A47D8 Win: 0x7B TcpLen: 20↵
```

Snort Rule

```
alert tcp any any -> any any (msg:"Advanced exploit detected"; \
sid:1000002; content:"backdoor"; reference:myserver,myrules; \
gid:1000001; rev:1; classtype:shellcode-detect; priority:100; \
metadata:meta data;)
```

Log Sample

```
[**] [1000001:1000002:1] Advanced exploit detected [**]↵
[Classification: Executable Code was Detected] [Priority: 100]↵
04/30-07:54:35.707783 192.168.255.110:46117 -> 172.25.212.204:80↵
TCP TTL:127 TOS:0x0 ID:14547 IpLen:20 DgmLen:435 DF↵
***AP*** Seq: 0x49649AA5 Ack: 0x5BC496C0 Win: 0x40 TcpLen: 20↵
[Xref => myserver myrules]
```

Example 280. Parsing Snort Logs

This configuration uses an `xm_multiline` extension module instance with a `HeaderLine` regular expression to parse the log entries. An `Exec` directive is also used to drop all empty lines.

In the Input module instance, another regular expression captures the parts of the message and adds corresponding fields to the event record. Additional information could be extracted also, such as Xref data, by adding `(.*)\s+(.*)\s+[\Xref => (.*)\]` to the expression and then `$Xref = $13;` below it.

Finally, the log entries are formatted as JSON with the `to_json()` procedure.

nxlog.conf

```

1 <Extension snort>
2   Module      xm_multiline
3   HeaderLine  /^[\*\*\*] \[ \S+ ] (.*) \[ \*\*\* ]/
4   Exec        if $raw_event =~ ^\s+$/ drop();
5 </Extension>
6
7 <Extension _json>
8   Module      xm_json
9 </Extension>
10
11 <Input in>
12   Module      im_file
13   File        "/var/log/snort/alert"
14   InputType   snort
15   <Exec>
16     if $raw_event =~ /(?x)^[\*\*\*]\[ \S+ ](.*)\[ \*\*\* ]\s+
17       (?:\[Classification:\[ \^\]]+\])\ )?
18       \[Priority:\[ (\d+)\]\s+
19       (\d\d).(\d\d)\--(\d\d:\d\d:\d\d.\d\d)
20       \ (\d+\.\d+\.\d+\.\d+):\?(\d+)?\ ->
21       \ (\d+\.\d+\.\d+\.\d+):\?(\d+)?\s+\ /
22   {
23     $EventName = $1;
24     $Classification = $2;
25     $Priority = $3;
26     $EventTime = parsedate(year(now()) + "-" + $4 + "-" + $5 + " " + $6);
27     $SourceIPAddress = $7;
28     $SourcePort = $8;
29     $DestinationIPAddress = $9;
30     $DestinationPort = $10;
31   }
32   </Exec>
33 </Input>
34
35 <Output out>
36   Module      om_file
37   File        "/var/log/nxlog_snort"
38   Exec        to_json();
39 </Output>
```

Output Sample

```
{
  "EventReceivedTime": "2014-05-05 09:08:58",
  "SourceModuleName": "in",
  "SourceModuleType": "im_file",
  "EventName": "Advanced exploit detected",
  "Classification": "Executable Code was Detected",
  "Priority": "100",
  "EventTime": "2014-04-30 07:54:35",
  "SourceIPAddress": "192.168.255.110",
  "SourcePort": "46117",
  "DestinationIPAddress": "172.25.212.204",
  "DestinationPort": "80"
}
```

Chapter 71. Splunk

Splunk is a software platform for data collection, indexing, searching, and visualization. NXLog can be configured as an agent for Splunk Enterprise, collecting and forwarding logs to the Splunk instance. Splunk can accept logs forwarded via [TCP](#), [UDP](#), [TLS](#), or [HTTP](#).

See the [Splunk Enterprise documentation](#) for more information about configuring and using Splunk.

71.1. Configuring TCP or UDP Collection

TCP or UDP log collection can be added from the web interface, however [SSL/TLS encryption](#) must be configured by editing configuration files.

NOTE

Although the UDP protocol is faster than TCP, it does not provide confirmation of packet delivery. TCP log collection is recommended by Splunk.

1. On the Splunk web interface, go to **Settings > Data inputs**.
2. Select **TCP** or **UDP** under the **Local inputs** section.
3. Click the **[New]** button.
4. Enter the port number to listen on. Click **[Next]**.

Configure this instance to listen on any TCP or UDP port to capture data sent over the network (such as syslog). [Learn More](#)

<input type="radio"/> TCP	<input type="radio"/> UDP
Port?	515
Example: 514	
Source name override?	optional hostport
Only accept connection from?	optional example: 10.1.2.3, !badhost.splunk.com, *.splunk.com

5. Select the relevant **Source type**, for example **linux_audit**. Splunk comes with many predefined source types; for less common log sources you could create your own source type. Then, choose the **App context** (Splunk instance folder) and the destination **Index**. Set the **Host** value as desired. Go to **[Review]**.

Input Settings

Optional set additional input parameters for this data input as follows:

Source type

The source type is one of the default fields that Splunk assigns to all incoming data. It tells Splunk what kind of data you've got, so that Splunk can format the data intelligently during indexing. And it's a way to categorize your data, so that you can search it easily.

Select
New

linux_audit ▾

App context

Application contexts are folders within a Splunk instance that contain configurations for a specific use case or domain of data. App contexts improve manageability of input and source type definitions. Splunk loads all app contexts based on precedence rules. [Learn More](#)

App Context
Search & Reporting (search) ▾

Host

When Splunk indexes data, each event receives a "host" value. The host value should be the name of the machine from which the event originates. The type of input you choose determines the available configuration options. [Learn More](#)

Method?
IP
DNS
Custo...

Index

Splunk stores incoming data as events in the selected index. Consider using a "sandbox" index as a destination if you have problems determining a source type for your data. A sandbox index lets you troubleshoot your configuration without impacting production indexes. You can always change this setting later. [Learn More](#)

Index
Default ▾
Create a new index

- After verifying the information on summary page, click the **[Submit]** button.

Example 281. Sending Logs via TCP or UDP

This configuration illustrates how to send linux audit logs to Splunk via TCP.

nxlog.conf

```

1 <Input in>
2   Module im_linuaudit
3   <Rules>
4     -w /etc/passwd -p wa -k passwd_changes
5   </Rules>
6 </Input>
7
8 <Output out>
9   Module om_tcp
10  Host    127.0.0.1
11  Port    515
12 </Output>
```

Or send logs via UDP.

nxlog.conf

```

1 <Output out>
2   Module om_udp
3   Host    127.0.0.1
4   Port    514
5 </Output>
```

71.2. Configuring TLS Collection

1. In order to generate certificates, issue the following commands from the server's console.

```
$ mkdir /opt/splunk/etc/certs
$ export OPENSSL_CONF=/opt/splunk/openssl/openssl.cnf
$ /opt/splunk/bin/genRootCA.sh -d /opt/splunk/etc/certs
$ /opt/splunk/bin/genSignedServerCert.sh -d /opt/splunk/etc/certs -n splunk -c splunk -p
```

NOTE The script will ask for a password to protect the key.

2. Go to the app's folder and edit the inputs file. For the **Search & Reporting** app, the path is `$SPLUNK_HOME/etc/apps/search/local/inputs.conf`. Add `[tcp-ssl]` and `[SSL]` stanzas.

inputs.conf

```
[tcp-ssl://10514]
disabled = false
sourcetype = <optional>

[SSL]
serverCert = /opt/splunk/etc/certs/splunk.pem
sslPassword = <The password provided in step 1>
requireClientCert = false
```

3. Edit the `$SPLUNK_HOME/etc/system/local/server.conf` file, adding a `sslRootCAPath` value to the `[sslConfig]` stanza.

server.conf

```
[sslConfig]
sslPassword = <Automatically generated>
sslRootCAPath = /opt/splunk/etc/certs/cacert.pem
```

4. Finally, restart Splunk in order to apply the new configuration.

```
$ $SPLUNK_HOME/bin/splunk restart splunkd
```

5. Setup can be tested with `netstat` or a similar command. If everything went correctly, the following output is produced.

```
$ netstat -an | grep :10514
tcp    0    0 0.0.0.0:10514    0.0.0.0:*      LISTEN
```

6. Copy the `cacert.pem` file from `$SPLUNK_HOME/etc/certs` to the NXLog certificate directory.

Example 282. Sending Logs via TLS

This configuration illustrates how to send a log file via a TLS-encrypted connection. The `AllowUntrusted` setting is required in order to accept a self-signed certificate.

nxlog.conf

```
1 <Output out>
2   Module          om_ssl
3   Host            127.0.0.1
4   Port            10514
5   CertFile        %CERTDIR%/cacert.pem
6   AllowUntrusted  TRUE
7 </Output>
```

71.3. Configuring HTTP Event Collection (HEC)

HTTP event collection in Splunk Enterprise is a way to gather events in JSON format, via HTTP/HTTPS. HEC is a stateless, high performance solution. It is also easy to scale with a load balancer. Furthermore it offers token-based authorization.

1. By default, HEC is disabled. In order to enable it, go to **Settings > Data inputs > HTTP Event Collector**. Click the **[Global Settings]** button (in the upper-right corner). On the configuration screen, in the **All Tokens** section, click the **[Enabled]** button. The **SSL** checkbox to enable HTTPS is checked by default; it is recommended to leave it enabled, otherwise the data will be sent in plain text and therefore could be eavesdropped. Click **[Save]**.

The screenshot shows the 'HTTP Event Collector' configuration page. At the top right, there are three green buttons: 'Global Settings', 'New Token', and another unlabeled one. A tooltip above the 'Global Settings' button states: 'All the tokens are currently disabled. They can be enabled in the Global Settings.' Below the buttons, there are filters for 'App: All' and 'filter', and a dropdown for '20 per page'. A table header row includes columns for 'Name', 'Token Value', 'Source Type', 'Index', and 'Status'. A message at the bottom left says 'No tokens found.'

2. Once HEC is enabled, click **[New Token]**. Enter a name for the token and click **[Next]**.
3. Select the **Source type** (or leave as **Automatic**), set the **App context** (Splunk folder to store the settings), and configure the **Index** list with indexes that are allowed to get the data from HEC. Set one of them as the default. Then click **[Review]**.

The screenshot shows the 'Add Data' wizard at the 'Input Settings' step. The progress bar is at 'Input Settings'. The 'Source type' section defines it as 'Automatic'. The 'App context' section shows 'Search & Reporting (search)'. The 'Index' section allows selecting allowed indexes: 'history', 'main', and 'summary'. One index is selected as the 'Default Index' ('main'). A note at the bottom says 'Select indexes that clients will be able to select from.'

4. After verifying the information on summary page, click the **[Submit]** button.
5. Finally, the token is created and its value is presented. It should be saved for later use.

6. The configuration can be tested with the following command.

```
$ curl -k https://<host>:8088/services/collector -H 'Authorization: Splunk <token>' -d '{"event":"test"}'
```

If everything went correctly, Splunk will respond that the test event was delivered.

```
{"text": "Success", "code": 0}
```

Example 283. Forwarding Logs to HEC via om_http

The **xm_json** and **om_http** modules can be used together to send the data to HEC.

nxlog.conf

```
1 <Extension json>
2     Module      xm_json
3 </Extension>
4
5 <Output out>
6     Module      om_http
7     URL         https://127.0.0.1:8088/services/collector
8     HTTPSCAFile %CERTDIR%/cacert.pem
9
10    # Use this directive ONLY for self-signed certificates
11    HTTPSAllowUntrusted TRUE
12
13    <Exec>
14        # Generate $raw_event in JSON format
15        to_json();
16
17        # Encapsulate the data in 'event' envelope;
18        # it is required to be compliant with HEC event format.
19        $raw_event = '{"event": ' + $raw_event + '}';
20        add_http_header("Authorization", "Splunk <YOUR TOKEN HERE>");
21    </Exec>
22 </Output>
```

Example 284. Forwarding Logs to HEC via Perl Script

The `om_perl` module is a very flexible solution for forwarding the logs to HEC. It allows for further event processing via the Perl script, before sending the data to Splunk.

NOTE The `om_perl` module is only available on the Linux platform.

nxlog.conf

```

1 define PLCODEDIR /opt/nxsec/libexec/nxlog/modules/output/perl
2
3 <Extension json>
4     Module      xm_json
5 </Extension>
6
7 <Output out>
8     Module      om_perl
9     PerlCode   %PLCODEDIR%/splunk-hec.pl
10    <Exec>
11        to_json();
12        $raw_event = '{"event": ' + $raw_event + '}';
13    </Exec>
14 </Output>
```

Following is the Perl code for sending the logs to HEC.

splunk-hec.pl

```

use warnings;
use FindBin;
use lib "$FindBin::Bin/../../../../src/modules/extension/perl";
use Log::Nxlog;
use LWP::UserAgent;
use IO::Socket::SSL qw();

sub write_data
{
    my $ua = LWP::UserAgent->new(
        #use these options ONLY for self-signed certificates
        ssl_opts => {
            SSL_verify_mode => IO::Socket::SSL::SSL_VERIFY_NONE,
            verify_hostname => 0,
        }
    );

    my $server_endpoint = "https://127.0.0.1:8088/services/collector";

    my $req = HTTP::Request->new(POST => $server_endpoint);
    $req->header('content-type' => 'application/json');
    $req->header('Authorization' => 'Splunk <YOUR TOKEN HERE>');

    my ($event) = @_;
    my $rawevt = Log::Nxlog::get_field($event, 'raw_event');

    $req->content($rawevt);

    $ua->request($req);
}
```

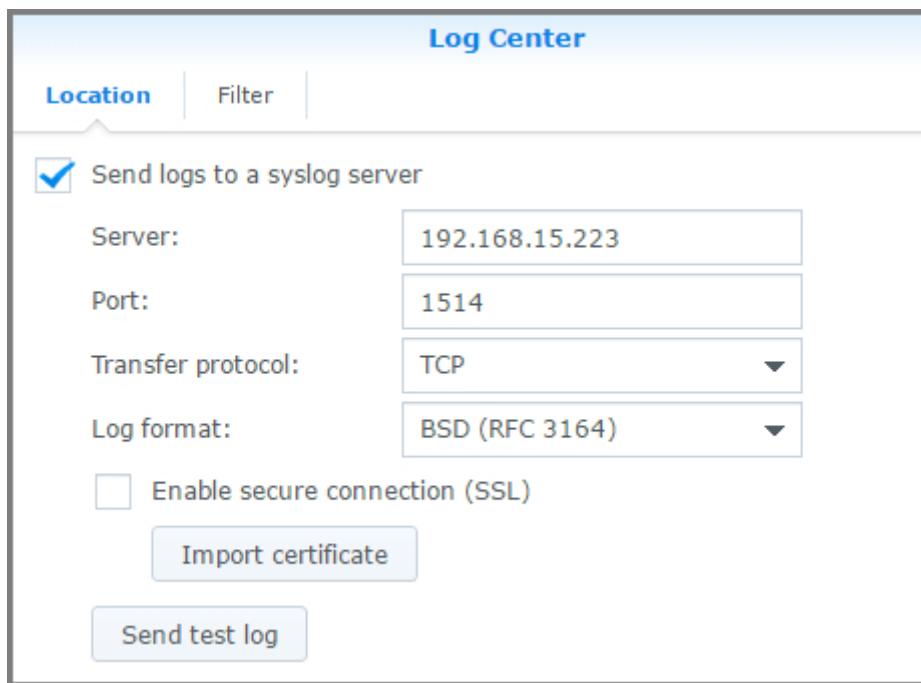
Chapter 72. Synology DiskStation

The Synology DiskStation is a Linux-based Network-attached storage (NAS) appliance. It runs syslog-ng and is capable of forwarding logs to a remote Syslog via UDP or TCP, including an option for SSL. Configuration is performed via the web interface.

NOTE

The steps below have been tested with DSM 5.2 and should work with newer versions as well.

1. Configure NXLog to receive log entries over the network and process them as Syslog (see the [TCP example](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from DiskStation device being configured.
3. Log in to the DiskStation web interface.
4. Go to **Log Center > Log Sending**.
5. Under the **Location** tab, specify the Syslog server, port, protocol, and log format. Enable and configure SSL if required.



6. Click **[Apply]**.

Example 285. Receiving DiskStation Logs via TCP

This configuration uses the `im_tcp` module to collect the DiskStation logs via TCP. A JSON output sample shows the resulting logs as received and processed by NXLog.

nxlog.conf

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module xm_json
7 </Extension>
8
9 <Input in>
10  Module im_tcp
11  Host 0.0.0.0
12  Port 1514
13  Exec parse_syslog();
14 </Input>
15
16 <Output out>
17  Module om_file
18  File  "/var/log/synology.log"
19  Exec to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.4.20",
  "EventReceivedTime": "2017-07-28 18:30:04",
  "SourceModuleName": "in_syslog_tcp",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "USER",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "DiskStation1",
  "EventTime": "2017-07-28 18:30:02",
  "Message": "Connection PWD\\sql_psqldw1:\tCIFS client [PWD\\sql_psqldw1] from [192.168.15.138(IP:192.168.15.138)] accessed the shared folder [db_backup]."
}
{
  "MessageSourceAddress": "192.168.4.20",
  "EventReceivedTime": "2017-07-28 18:29:48",
  "SourceModuleName": "in_syslog_tcp",
  "SourceModuleType": "im_tcp",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "USER",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "DiskStation1",
  "EventTime": "2017-07-28 18:29:56",
  "Message": "System Test message from Synology Syslog Client from (0.240.175.244)"
}
```

Chapter 73. Syslog

NXLog can be configured to collect or generate log entries in the various Syslog formats. This section describes the various Syslog protocols and discusses how to use them with NXLog.

73.1. BSD Syslog (RFC 3164)

The original Syslog was written for the Sendmail project in the 1980s. It was later adopted by many other applications and implemented across many operating systems, and became the standard logging system for Unix-style systems. There was no authoritative publication about Syslog until 2001, when the Internet Engineering Task Force (IETF) published informational RFC 3164, which described the "observed behavior" among implementations. Today, modern implementations follow RFC 3164, which attempts to accommodate the many older implementations; still it must be acknowledged that many undocumented variations exist among the BSD Syslog protocols as implemented by various applications and devices.

Log Sample

```
<30>Nov 21 11:40:27 myserver sshd[26459]: Accepted publickey for john from 192.168.1.1 port 41193  
ssh2<
```

BSD Syslog defines both the log entry format and the transport. The message format is free-form, allowing for the payload to be JSON or another structured data format.

73.1.1. BSD Syslog Format

BSD Syslog uses a simple format, comprised of three parts.

Base BSD Syslog Format

```
<PRI>HEADER MSG<
```

NOTE

While this is the common and recommended format for a BSD Syslog message, there are no set requirements and a device may send a BSD Syslog log message containing only a free-form message, without PRI or HEADER parts.

The PRI part, or "priority", is calculated from the facility and severity codes. The facility code indicates the type of program that generated the message, and the severity code indicates the severity of the message (see the [Syslog Facilities](#) and [Syslog Severities](#) tables below). The priority code is calculated by multiplying the facility code by eight and then adding the severity code.

NOTE

The PRI part is not written to file by many Syslog loggers. In that case, each log entry begins with the HEADER.

The HEADER part contains two fields: TIMESTAMP and HOSTNAME. The TIMESTAMP provides the local time when the message was generated in **Mmm dd hh:mm:ss** format, with no year or time zone specified; the HOSTNAME is the name of the host where the message was generated.

The MSG part contains two fields: TAG and CONTENT. The TAG is the name of the program or process that generated the message, and contains only alphanumeric characters. Any other character will represent the beginning of the CONTENT field. The CONTENT field often contains the process ID enclosed by brackets ([]), a colon (:), a space, and then the actual message. In the [log sample](#) above, the MSG part begins with **myserver sshd[26459]: Accepted publickey**; in this case, the TAG is **ssh** and the CONTENT field begins with **[26459]**. The CONTENT field can contain only ASCII printable characters (32-126).

Fields Commonly Used in the BSD Syslog Format

```
<PRI>TIMESTAMP HOSTNAME TAG[PID]: MESSAGE<
```

Table 56. Syslog Facilities

Facility Code	Description
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	scheduling daemon
16	local use 0 (local0)
17	local use 1 (local1)
18	local use 2 (local2)
19	local use 3 (local3)
20	local use 4 (local4)
21	local use 5 (local5)
22	local use 6 (local6)
23	local use 7 (local7)

Table 57. Syslog Severities

Severity Code	Description
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

73.1.2. BSD Syslog Transport

According to RFC 3164, the BSD Syslog protocol uses UDP as its transport layer. Each UDP packet carries a single log entry. BSD Syslog implementations often also support plain TCP and TLS transports, though these are not covered by RFC 3164.

73.1.3. Disadvantages of BSD Syslog

There are several disadvantages associated with the BSD Syslog protocol.

- The transport defined by RFC 3164 uses UDP and provides no mechanism to ensure reliable delivery, integrity, or confidentiality of log messages.
- Many undocumented variations exist among implementations.
- The timestamp indicates neither the year nor the timezone, and does not provide precision greater than the second.
- The PRI field (and therefore the facility and severity codes) are not retained by many Syslog loggers when writing to log files.
- The entire length of the log entry is limited to 1024 bytes.
- Only ASCII characters 32-126 are allowed, no Unicode or line breaks.

73.2. IETF Syslog (RFCs 5424-5426)

In 2009, the IETF released RFCs 5424, 5425, and 5426 as "Proposed Standard"s intended to replace the "legacy" BSD Syslog. RFC 5425 includes a timestamp with year, timezone, and fractional seconds; provides a "structured data" field for key-value pairs; and offers UTF-8 encoding. RFC 5425 defines the use of TLS transport and supports multi-line log messages. RFC 5426 describes the use of UDP transport.

Log Sample

```
<165>1 2003-10-11T22:14:15.003Z mymachine.example.com evntslog - ID47 [exampleSDID@32473 iut="3"
eventSource="Application" eventID="1011"] An application event log entry...«
```

73.2.1. IETF Syslog Format

IETF Syslog uses a base format similar to that of BSD Syslog.

Base IETF Syslog Format

```
HEADER STRUCTURED-DATA MSG«
```

The HEADER part contains seven fields.

- PRI: message priority (same as BSD Syslog)
- VERSION: Syslog format version (always "1" for RFC 5424 logs)
- TIMESTAMP: derived from RFC 3339 (**YYYY-MM-DDTHH:MM:SS.000000Z**, or with the time zone specified)
- HOSTNAME
- APP-NAME: device or application that generated the message
- PROCID: ID of the process that generated the message
- MSGID: message type (for example, "TCPIN" for incoming TCP traffic and "TCPOUT" for outgoing)

NOTE

The PRI field is not written to file by many Syslog loggers. In that case, each log entry begins with the VERSION field.

The STRUCTURED-DATA part is optional. If it is omitted, then a hyphen acts as a placeholder. Otherwise, it is

surrounded by brackets. It contains an ID of the block and a list of space-separated "key=value" pairs.

The MSG part is optional and contains a free-form, single-line message. If the message is encoded in UTF-8, then it may be preceded by a Unicode byte order mark (BOM).

Fields in the IETF Syslog Format

```
<PRI>VERSION TIMESTAMP HOSTNAME APP-NAME PROCID MSGID [SD-ID STRUCTURED-DATA] MESSAGE↔
```

73.2.2. IETF Syslog Transport

IETF Syslog can use UDP, plain TCP, or TLS transport. UDP transport is described by RFC 5426, TLS transport by RFC 5425.

RFC 5425 also documents the octet-framing method that is used for TLS transport and provides support for multi-line messages. Octet-framing can be used with plain TCP also, TLS is not required. The message length is pre-pended as in the following example which shows the raw data that is sent over TCP/TLS.

Log Sample With Octet Framing

```
101 <13>1 2012-01-01T17:15:52.873750+01:00 myhost - - - [NXLOG@14506 TestField="test value"] test  
message↔
```

In practice IETF Syslog is commonly transferred without octet-framing over TCP or TLS. In this case the newline (\n) character is used as the record separator, similarly to how BSD Syslog is transferred over TCP or TLS.

73.3. Collecting and Parsing Syslog

NXLog can be configured to collect Syslog logs by:

- reading Syslog files written by another local Syslog agent,
- accepting Syslog via the local `/dev/log` Unix domain socket, or
- accepting Syslog over the network (via UDP, TCP, or TLS).

73.3.1. Reading Syslog Log Files

Configuring NXLog to read Syslog from file allows another local Syslog agent to continue its logging operations as before. Note that NXLog will likely not have access to the facility and severity codes because most Syslog loggers do not write the PRI field to log files.

Make sure NXLog has permission to read log files in `/var/log`. See [Reading Rsyslog Log Files](#) for more information.

Example 286. Reading Syslog From File

This configuration reads log messages from file and parses them using the [parse_syslog\(\)](#) procedure.

nxlog.conf

```
1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input in>
6   Module im_file
7   File   '/var/log/messages'
8   Exec   parse_syslog();
9 </Input>
```

NOTE

The [parse_syslog\(\)](#) procedure parses the log entry as either BSD or IETF format (the [parse_syslog_bsd\(\)](#) and [parse_syslog_ietf\(\)](#) procedures can be used alternatively).

73.3.2. Accepting Syslog via /dev/log

Many applications support logging by sending log messages to the [/dev/log](#) Unix domain socket. It is the responsibility of the system logger to accept these messages and then store them as configured. NXLog can be configured to directly accept logs that are sent to the [/dev/log](#) Unix domain socket, in place of the stock Syslog logger.

1. Configure NXLog (see the [example](#) below).
2. Disable the stock Syslog agent's collection of [/dev/log](#) messages, if necessary. See also [Replacing Rsyslog](#). Either
 - ° disable the service entirely (for example, `systemctl --now disable rsyslogd`) or
 - ° modify the configuration to disable reading from [/dev/log](#) (for example, remove `$ModLoad imuxsock` from `/etc/rsyslog.conf` and restart Rsyslog).
3. Restart NXLog.

Example 287. Reading From /dev/log

With this configuration, NXLog uses the `im_uds` module to read messages from `/dev/log`, and the `parse_syslog()` procedure to parse them.

WARNING

`FlowControl` should be disabled for collecting from `/dev/log`. Otherwise the `syslog()` system call will block if the Output queue becomes full, resulting in an unresponsive system.

nxlog.conf

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input in>
6   Module      im_uds
7   UDS        /dev/log
8   FlowControl FALSE
9   Exec       parse_syslog();
10 </Input>
```

73.3.3. Accepting Syslog via UDP, TCP, or TLS

NXLog can be configured to listen on a port and collect Syslog over the network. A port can be used to receive messages with UDP, TCP, or TLS transport. The local Syslog agent may already be configured to listen on port 514 for UDP log messages from local applications.

1. Configure NXLog with `im_udp`, `im_tcp`, or `im_ssl`. See the examples below.
2. For NXLog to listen for messages on port 514, the local Syslog agent must not be listening on that port. It may be necessary to either
 - disable the service entirely (for example, `systemctl disable rsyslogd`) or
 - modify the configuration to disable listening on port 514 (for example, remove `input(type="imudp" port="514")` from `/etc/rsyslog.conf` and restart Rsyslog).
3. Restart NXLog.

Example 288. Receiving Syslog via UDP

This configuration accepts either BSD or IETF Syslog from the local system only, via UDP.

WARNING

The UDP transport can lose log entries and is therefore not recommended for receiving logs over the network.

nxlog.conf

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input in>
6   Module      im_udp
7   Host        localhost
8   Port        514
9   Exec       parse_syslog();
10 </Input>
```

Example 289. Receiving Syslog via TCP

This configuration accepts either BSD or IETF Syslog via TCP, without supporting octet-framing.

nxlog.conf

```
1 <Extension _syslog>
2     Module    xm_syslog
3 </Extension>
4
5 <Input in>
6     Module   im_tcp
7     Host     0.0.0.0
8     Port     1514
9     Exec     parse_syslog();
10 </Input>
```

Example 290. Receiving IETF Syslog via TCP With Octet-Framing

This configuration accepts IETF Syslog via TCP, with support for octet-framing.

NOTE

Though this is for plain TCP, the [Syslog_TLS](#) directive is required because it refers to the octet-framing method described by RFC 5425.

nxlog.conf

```
1 <Extension _syslog>
2     Module    xm_syslog
3 </Extension>
4
5 <Input in>
6     Module   im_tcp
7     Host     0.0.0.0
8     Port     1514
9     InputType Syslog_TLS
10    Exec     parse_syslog_ietf();
11 </Input>
```

Example 291. Receiving IETF Syslog via TLS

This configuration accepts IETF Syslog via TLS, with support for octet-framing.

nxlog.conf

```
1 <Extension _syslog>
2     Module    xm_syslog
3 </Extension>
4
5 <Input in>
6     Module   im_ssl
7     Host     0.0.0.0
8     Port     6514
9     CAFile   %CERTDIR%/ca.pem
10    CertFile  %CERTDIR%/client-cert.pem
11    CertKeyFile %CERTDIR%/client-key.pem
12    InputType Syslog_TLS
13    Exec     parse_syslog_ietf();
14 </Input>
```

73.4. Generating Syslog

NXLog can be configured to generate BSD or IETF Syslog and:

- write it to file,
- send it to the local syslog daemon via the `/dev/log` Unix domain socket, or
- forward it to another destination over the network (via UDP, TCP, or TLS).

In each case, the `to_syslog_bsd()` and `to_syslog_ietf()` procedures are used to generate the `$raw_event` field from the corresponding `fields` in the event record.

73.4.1. Writing Syslog to File

The `om_file` module is used to write logs to file.

Example 292. Writing BSD Syslog to File

This configuration write logs to the specified file in the BSD Syslog format.

`nxlog.conf`

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Output out>
6   Module om_file
7   File   "/var/log/syslog"
8   Exec   to_syslog_bsd();
9 </Output>
```

NXLog can be configured to write BSD Syslog to a file without the PRI part, emulating traditional Syslog implementations.

Example 293. Writing BSD Syslog Without the PRI

This configuration includes a regular expression for removing the PRI part from the `$raw_event` field after it is generated by the `to_syslog_bsd()` procedure.

`nxlog.conf`

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Output out>
6   Module om_file
7   File   "/var/log/syslog"
8   Exec   to_syslog_bsd(); $raw_event =~ s/^<\d+>//;
9 </Output>
```

73.4.2. Sending Syslog to the Local Syslog Daemon via `/dev/log`

The `om_uds` module can be used for sending logs to a Unix domain socket.

Example 294. Sending Syslog to /dev/log

This configuration sends BSD Syslog to the Syslog daemon via `/dev/log`.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Output out>
6     Module om_uds
7     UDS    /dev/log
8     Exec   to_syslog_bsd();
9 </Output>
```

73.4.3. Sending Syslog to a Remote Logger via UDP, TCP, or TLS

The `om_udp`, `om_tcp`, and `om_ssl` modules can be used for sending Syslog over the network.

Example 295. Forwarding BSD Syslog via UDP

This configuration sends logs in BSD Syslog format to the specified host, via UDP port 514.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Output out>
6     Module om_udp
7     Host   192.168.1.1
8     Port   514
9     Exec   to_syslog_bsd();
10 </Output>
```

Example 296. Forwarding BSD Syslog via TCP

This configuration sends logs in BSD format to the specified host, via TCP port 1514.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Output out>
6     Module om_tcp
7     Host   192.168.1.1
8     Port   1514
9     Exec   to_syslog_bsd();
10 </Output>
```

Example 297. Forwarding IETF Syslog via TLS

With this configuration, NXLog sends logs in IETF format to the specified host, via port 6514.

nxlog.conf

```
1 <Extension _syslog>
2     Module      xm_syslog
3 </Extension>
4
5 <Output out>
6     Module      om_ssl
7     Host        192.168.1.1
8     Port        6514
9     CAFile      %CERTDIR%/ca.pem
10    CertFile    %CERTDIR%/client-cert.pem
11    CertKeyFile %CERTDIR%/client-key.pem
12    OutputType  Syslog_TLS
13    Exec        to_syslog_ietf();
14 </Output>
```

NOTE

The **OutputType Syslog_TLS** directive is necessary if octet-framing is required. The name was chosen to refer to the octet-framing method described by RFC 5425.

73.5. Extending Syslog

BSD Syslog uses a free-form message field, and does not provide a standard way to include key-value pairs in log messages. This section documents ways that structured data has been implemented using BSD Syslog as transport.

73.5.1. IETF Syslog Structured-Data

The Structured-Data part of the IETF Syslog format, as documented above, provides a syntax for key-value pairs.

Log Sample

```
<13>1 2016-10-13T14:23:11.000000-06:00 myserver - - - [NXLOG@14506 Purpose="test"] This is a test
message.←
```

NXLog can parse IETF Syslog with the [parse_syslog\(\)](#) procedure provided by the *xm_syslog* extension module.

Example 298. Parsing IETF Syslog With Structured-Data

With this configuration, NXLog will parse the input IETF Syslog format from file, convert it to JSON, and output the result to file.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Extension _syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input in>
10    Module im_file
11    File   '/var/log/messages'
12    Exec   parse_syslog();
13 </Input>
14
15 <Output out>
16    Module om_file
17    File   '/var/log/json'
18    Exec   to_json();
19 </Output>
20
21 <Route r>
22     Path   in => out
23 </Route>
```

Output Sample

```
{
  "EventReceivedTime": "2016-10-13 15:23:12",
  "SourceModuleName": "in",
  "SourceModuleType": "im_file",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "USER",
  "SyslogSeverityValue": 5,
  "SyslogSeverity": "NOTICE",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventTime": "2016-10-13 15:23:11",
  "Hostname": "myserver",
  "Purpose": "test",
  "Message": "This is a test log message."
}
```

NXLog can also generate IETF Syslog with a Structured-Data part, using the [to_syslog_ietf\(\)](#) procedure provided by the *xm_syslog* extension module.

Example 299. Generating IETF Syslog With Structured-Data

With the following configuration, NXLog will parse the input JSON from file, convert it to IETF Syslog format, and output the result to file.

nxlog.conf

```

1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 <Extension _syslog>
6   Module xm_syslog
7 </Extension>
8
9 <Input in>
10  Module im_file
11  File  '/var/log/json'
12  Exec  parse_json();
13 </Input>
14
15 <Output out>
16  Module om_file
17  File  '/var/log/ietf'
18  Exec  to_syslog_ietf();
19 </Output>
20
21 <Route r>
22  Path  in => out
23 </Route>
```

Input Sample

```
{
  "EventTime": "2016-09-13 11:23:11",
  "Hostname": "myserver",
  "Purpose": "test",
  "Message": "This is a test log message."
}
```

Output Sample

```
<13>1 2016-09-13T11:23:11.000000-05:00 myserver - - - [NXLOG@14506 EventReceivedTime="2016-09-13 11:23:12" SourceModuleName="in" SourceModuleType="im_file" Purpose="test] This is a test log message.←
```

73.5.2. JSON over Syslog

JSON has become popular recently to transfer structured data. For compatibility with Syslog devices it is common practice to encapsulate JSON in Syslog. NXLog can generate JSON with the [to_json\(\)](#) procedure function provided by the *xm_json* extension module.

Example 300. Generating JSON with Syslog Header

With the following configuration, NXLog will read the Windows Event Log, convert it to JSON format, add a Syslog header, and send the logs via UDP to a Syslog agent. NXLog log messages are also included (via the [im_internal](#) module).

nxlog.conf

```
1 <Extension _json>
2     Module      xm_json
3 </Extension>
4
5 <Extension _syslog>
6     Module      xm_syslog
7 </Extension>
8
9 <Input internal>
10    Module     im_internal
11 </Input>
12
13 <Input eventlog>
14    Module     im_msvisalog
15 </Input>
16
17 <Output out>
18    Module     om_udp
19    Host       192.168.1.1
20    Port       514
21    Exec       $Message = to_json(); to_syslog_bsd();
22 </Output>
23
24 <Route r>
25    Path       internal, eventlog => out
26 </Route>
```

NOTE

If Syslog compatibility is not a concern, JSON can be transported without the Syslog header (omit the [to_syslog_bsd\(\)](#) procedure).

73.5.3. Other Syslog Extensions

See also these formats, which extend BSD Syslog by using the free-form message field to contain key-value pairs.

- [ArcSight Common Event Format \(CEF\)](#)
- [Common Event Expression \(CEE\)](#)
- [Log Event Extended Format \(LEEF\)](#)
- [Snare](#)

Chapter 74. Sysmon

NXLog can be configured to capture and process audit logs generated by the Sysinternals [Sysmon](#) utility. Sysmon is a Windows system service and device driver that logs system activity to the Windows EventLog. Supported events include (but are not limited to):

- process creation and the full command line used,
- loading of system drivers,
- network connections, and
- modification of file creation timestamps.

On Windows Vista and higher, Sysmon's events are stored in "Applications and Services Logs/Microsoft/Windows/Sysmon/Operational". On older systems, events are written to the System event log.

74.1. Sysmon Events

When Sysmon generates EventLog data, it encodes details of the event into the **EventData** tag of the EventLog record.

Example Sysmon EventLog Entry

```
...
<EventData>
  <Data Name="UtcTime">2015.04.27. 13:23</Data>
  <Data Name="ProcessGuid">{00000000-3862-553E-0000-001051D40527}</Data>
  <Data Name="ProcessId">25848</Data>
  <Data Name="Image">c:\Program Files (x86)\nxlog\nxlog.exe</Data>
  <Data Name="CommandLine">"c:\Program Files (x86)\nxlog\nxlog.exe" -f</Data>
  <Data Name="User">WIN-OUNNPISDHIG\Administrator</Data>
  <Data Name="LogonGuid">{00000000-568E-5453-0000-0020D5ED0400}</Data>
  <Data Name="LogonId">0x4edd5</Data>
  <Data Name="TerminalSessionId">2</Data>
  <Data Name="IntegrityLevel">High</Data>
  <Data Name="HashType">SHA1</Data>
  <Data Name="Hash">1DCE4B0F24C40473Ce7B2C57EB4F7E9E3E14BF94</Data>
  <Data Name="ParentProcessGuid">{00000000-3862-553E-0000-001088D30527}</Data>
  <Data Name="ParentProcessId">26544</Data>
  <Data Name="ParentImage">C:\msys\1.0\bin\sh.exe</Data>
  <Data Name="ParentCommandLine">C:\msys\1.0\bin\sh.exe</Data>
</EventData>
</Event>
```

NXLog will automatically parse the **Data** tags so the values are available as fields in the NXLog event records. See the JSON output below.

74.2. Collecting Sysmon Events

NXLog can be configured to collect the Sysmon audit log data. It can then be forwarded to a log analytics system to allow identification of malicious or anomalous activity.

1. Download the [Sysmon](#) utility.
2. Configure and install the Sysmon service.

Example 301. Configuring Sysmon

The following example configuration file:

- logs network connections,
- logs all driver activity except for Microsoft or Windows core drivers,
- excludes process terminations,
- excludes file creation timestamps, and
- excludes network connections of a particular process and port.

Sysmon XML Configuration

```
<Sysmon schemaversion="1.0">
<Configuration>
    <!-- Capture MD5 Hashes -->
    <Hashing>MD5</Hashing>
    <!-- Enable network logging -->
    <Network />
</Configuration>
<Rules>
    <!-- Log all drivers except if the signature -->
    <!-- contains Microsoft or Windows -->
    <DriverLoad default="include">
        <Signature condition="contains">microsoft</Signature>
        <Signature condition="contains">windows</Signature>
    </DriverLoad>
    <!-- Do not log process termination -->
    <ProcessTerminate />
    <!-- Exclude certain processes that cause high event volumes -->
    <ProcessCreate default="include">
        <Image condition="contains">noisyprogram.exe</Image>
    </ProcessCreate>
    <!-- Do not log file creation time stamps -->
    <FileCreateTime />
    <!-- Do not log network connections of a certain process or port -->
    <NetworkConnect default="include">
        <Image condition="contains">someapp.exe</Image>
        <DestinationPort>4041</DestinationPort>
    </NetworkConnect>
</Rules>
</Sysmon>
```

3. Configure NXLog.

Example 302. Collecting Sysmon Logs

This configuration will gather Syslog events from the EventLog and write them to file in JSON format.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Input in>
6     Module im_msvisatalog
7     <QueryXML>
8         <QueryList>
9             <Query Id="0">
10                <Select Path="Microsoft-Windows-Sysmon/Operational">*</Select>
11            </Query>
12        </QueryList>
13    </QueryXML>
14 </Input>
15
16 <Output out>
17     Module om_file
18     File   'C:\test\sysmon.json'
19     Exec   to_json();
20 </Output>
21
22 <Route r>
23     Path   in => out
24 </Route>
```

Output Sample

```
{  
    "EventTime": "2015-04-27 15:23:46",  
    "Hostname": "WIN-OUNNPISDHIG",  
    "Keywords": -9223372036854776000,  
    "EventType": "INFO",  
    "SeverityValue": 2,  
    "Severity": "INFO",  
    "EventID": 1,  
    "SourceName": "Microsoft-Windows-Sysmon",  
    "ProviderGuid": "{5770385F-C22A-43E0-BF4C-06F5698FFBD9}",  
    "Version": 3,  
    "Task": 1,  
    "OpcodeValue": 0,  
    "RecordNumber": 2335906,  
    "ProcessID": 1680,  
    "ThreadID": 1728,  
    "Channel": "Microsoft-Windows-Sysmon/Operational",  
    "Domain": "NT AUTHORITY",  
    "AccountName": "SYSTEM",  
    "UserID": "SYSTEM",  
    "AccountType": "Well Known Group",  
    "Message": "Process Create:\r\nUtcTime: 2015.04.27. 13:23\r\nProcessGuid: {00000000-3862-553E-0000-001051D40527}\r\nProcessId: 25848\r\nImage: c:\\Program Files (x86)\\nxlog\\nxlog.exe\r\nCommandLine: \"c:\\Program Files (x86)\\nxlog\\nxlog.exe\" -f\r\nUser: WIN-OUNNPISDHIG\\Administrator\r\nLogonGuid: {00000000-568E-5453-0000-0020D5ED0400}\r\nLogonId: 0x4edd5\r\nTerminalSessionId: 2\r\nIntegrityLevel: High\r\nHashType: SHA1\r\nHash: 1DCE4B0F24C40473CE7B2C57EB4F7E9E3E14BF94\r\nParentProcessGuid: {00000000-3862-553E-0000-001088D30527}\r\nParentProcessId: 26544\r\nParentImage: C:\\msys\\1.0\\bin\\sh.exe\r\nParentCommandLine: C:\\msys\\1.0\\bin\\sh.exe",  
    "Opcode": "Info",  
    "UtcTime": "2015.04.27. 13:23",  
    "ProcessGuid": "{00000000-3862-553E-0000-001051D40527}",  
    "Image": "c:\\Program Files (x86)\\nxlog\\nxlog.exe",  
    "CommandLine": "\"c:\\Program Files (x86)\\nxlog\\nxlog.exe\" -f",  
    "User": "WIN-OUNNPISDHIG\\Administrator",  
    "LogonGuid": "{00000000-568E-5453-0000-0020D5ED0400}",  
    "LogonId": "0x4edd5",  
    "TerminalSessionId": "2",  
    "IntegrityLevel": "High",  
    "HashType": "SHA1",  
    "Hash": "1DCE4B0F24C40473CE7B2C57EB4F7E9E3E14BF94",  
    "ParentProcessGuid": "{00000000-3862-553E-0000-001088D30527}",  
    "ParentProcessId": "26544",  
    "ParentImage": "C:\\msys\\1.0\\bin\\sh.exe",  
    "ParentCommandLine": "C:\\msys\\1.0\\bin\\sh.exe",  
    "EventReceivedTime": "2015-04-27 15:23:47",  
    "SourceModuleName": "in",  
    "SourceModuleType": "im_msvistalog"  
}
```

Example 303. Forwarding Sysmon Logs

To send the logs to a remote server in JSON format with a BSD Syslog header, instead use an Output configuration like this.

nxlog.conf

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6   Module xm_json
7 </Extension>
8
9 <Output out>
10  Module om_tcp
11    Host 10.0.0.1
12    Port 1514
13    Exec to_json(); $Message = $raw_event; to_syslog_bsd();
14 </Output>
```

74.3. Advanced Filtering Options

Some scenarios require more advanced filtering of Sysmon logs in order to achieve more useful results. There are three main ways to filter Sysmon logs.

Sysmon configuration

This method involves adjusting the Sysmon configuration to avoid logging unwanted events. This can be done with filtering tags. See the [Sysmon](#) page for details of the available tags. This method is the most efficient because it avoids creating the unwanted log entries in the first place.

im_msvistalog XPath Query directive

This method uses the [Query](#) directive in the [im_msvistalog](#) module. The configuration specifies an XPath query that NXLog passes to the Windows EventLog API to limit the entries. Because this method restricts the number of entries that reach NXLog, it is a fairly efficient way to filter logs. The following example shows a query that preserves only operational logs with an ID of 1.

nxlog.conf

```

1 <QueryXML>
2   <QueryList>
3     <Query Id="0">
4       <Select Path="Microsoft-Windows-Sysmon/Operational">
5         *[System[(EventID='1')]]
6       </Select>
7     </Query>
8   </QueryList>
9 </QueryXML>
```

NXLog language

The built-in filtering capabilities of NXLog can be used, and may be easier to write than the XML query syntax provided by the EventLog API. The following example shows a query that preserves only operational logs with an ID of 1, like the XPath example above.

```

1 if not ($Channel == 'Microsoft-Windows-Sysmon' and
2      $EventID == 1) drop();
```

This example removes all event records regarding HTTP network connections to a particular server.

```
1 if $SourceName == 'Microsoft-Windows-Sysmon' and  
2     $DestinationPort == 80 and  
3     $DestinationIp == 10.0.0.1 drop();
```

Chapter 75. Ubiquiti UniFi

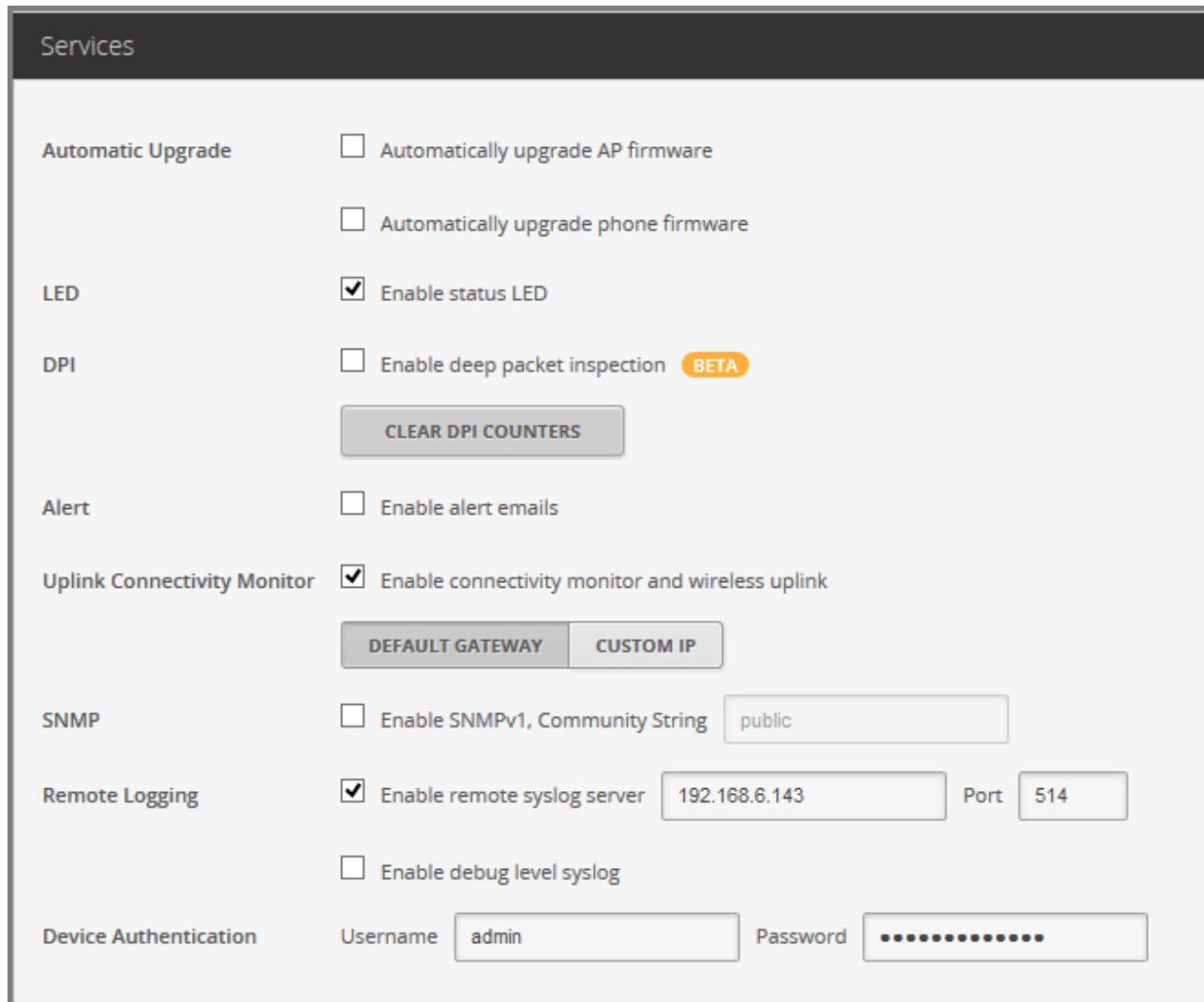
Ubiquiti UniFi is an enterprise solution for managing wireless networks. The UniFi infrastructure is managed by the UniFi Controller, which can be configured to send logs to a remote Syslog server via UDP. As a central management point, it will make sure that logs from all access points, including client authentication messages, are logged to the Syslog server.

More information about configuring the UniFi Controller can be found in the corresponding [user guide](#).

NOTE

The steps below have been tested with UniFi Controller v4 and should work for other versions also.

1. Configure NXLog for receiving Syslog log entries via UDP (see the [examples](#) below). Then restart NXLog.
2. Make sure the NXLog agent is accessible from the server with the Controller software.
3. Log in to the Controller's web interface.
4. Go to **Settings > Site**.
5. Select **Enable remote syslog server** and specify the IP address and UDP port that the NXLog agent is listening on. If necessary, also select **Enable debug level syslog**. Then click [**Apply**].



By default, the UniFi Controller sends a lot of low level information which may complicate field extraction if additional intelligence is required. The Syslog level can be adjusted individually for each access point from the Controller server by changing the `syslog.level` value in the `system.cfg` file. The location of this file varies depending on the host operating system. If the Controller software is running on Windows, the file can be found

under `C:\Ubiquiti UniFi\data\devices\uap\<AP_MAC_ADDRESS>`

Unfortunately, once configured with remote Syslog address, the Controller only sends log messages that originate from access points. The Controller's own log is located on the server where it is installed. The location of this file depends on the host operating system, on Windows it can be found at `C:\Ubiquiti UniFi\logs\server.log`. If needed, this file can be parsed with the `om_file` module.

Example 304. Collecting UniFi Logs From the Controller

This example shows UniFi logs as received and processed by NXLog.

`nxlog.conf`

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in_syslog_udp>
10    Module im_udp
11    Host   0.0.0.0
12    Port   514
13    Exec   parse_syslog();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File   "/var/log/unifi.log"
19    Exec   to_json();
20 </Output>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.10.147",
  "EventReceivedTime": "2017-04-27 19:38:55",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 3,
  "SyslogFacility": "DAEMON",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "192.168.10.147",
  "EventTime": "2017-04-27 19:40:44",
  "Message": "(\"U7P,0418d6809ce2,v3.7.11.5131\") hostapd: ath4: STA 34:02:86:45:8e:e0 IEEE
802.11: disassociated"
}
```

Example 305. Extracting Additional Fields

Additional fields can be extracted from the Syslog messages with a configuration like the one below.

nxlog.conf

```

1 <Input in_syslog_udp>
2   Module im_udp
3   Host   0.0.0.0
4   Port   514
5   <Exec>
6     parse_syslog();
7     if $Message =~ / ([a-z]*) (.*)$/
8     {
9       $UFProcess = $1;
10      $UFMessage = $2;
11      if $UFMessage =~ /^[a-z0-9]* (.*)$/
12      {
13        $UFSbsys = $1;
14        $UFMessage = $2;
15        if $UFMessage =~ ^STA (.*) ([A-Z0-9. ]*) (.*)$/
16        {
17          $UFMac = $1;
18          $UFProto = $2;
19          $UFMessage = $3;
20        }
21      }
22    }
23  </Exec>
24 </Input>
```

Output Sample

```
{
  "MessageSourceAddress": "192.168.10.149",
  "EventReceivedTime": "2017-05-01 20:30:13",
  "SourceModuleName": "in_syslog_udp",
  "SourceModuleType": "im_udp",
  "SyslogFacilityValue": 3,
  "SyslogFacility": "DAEMON",
  "SyslogSeverityValue": 6,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "192.168.10.149",
  "EventTime": "2017-05-01 20:32:11",
  "Message": "(\"U7P_0418d6809b78,v3.7.11.5131\") hostapd: ath2: STA 80:19:34:97:62:a6 RADIUS: stopped accounting session 5907CFDD-00000002",
  "UFProcess": "hostapd",
  "UFSbsys": "ath2",
  "UFMac": "80:19:34:97:62:a6",
  "UFProto": "RADIUS",
  "UFMessage": "stopped accounting session 5907CFDD-00000002"
}
```

Chapter 76. VMware vCenter

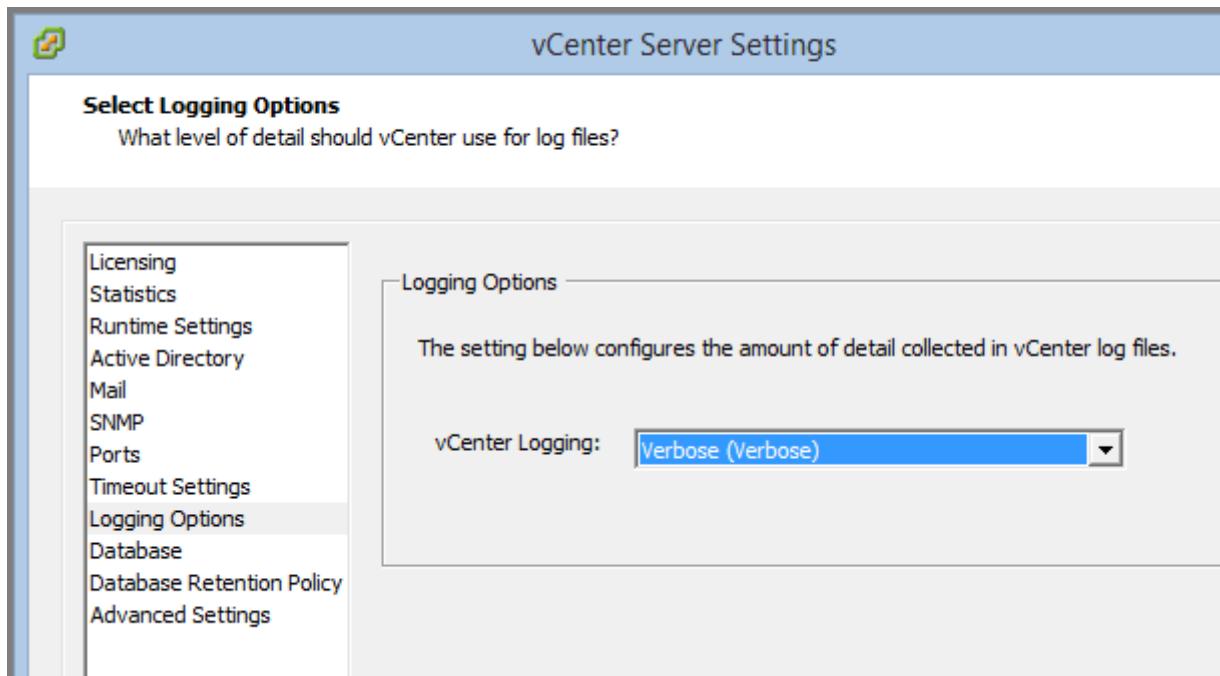
NXLog can be used to capture and process logs from VMware™ vCenter™. This guide explains how to do this with vCenter 5.5 installed on Windows™ Server 2008 R2.

vCenter logs can be processed in two ways.

- NXLog can be installed directly on the vCenter host machine and configured to collect all logs locally. This method provides more feedback and more detailed logs, and is the recommended method. See [Local vCenter Logging](#).
- Alternatively, vCenter logs can be collected remotely using the vCenter web SDK. This option is less flexible, but may be the only feasible option in certain environments due to security restrictions. See [Remote vCenter Logging](#).

76.1. Local vCenter Logging

1. Install NXLog on the vCenter host machine.
2. Log in to the vCenter client.
3. Open **Administration > vCenter Server Settings**, select **Logging Options** from the list on the left, and set **vCenter Logging** to **Verbose (Verbose)**.



4. Click **[OK]** to save your changes. vCenter will now start writing detailed logs. The location of the logs depends on the version of vCenter you are running.
 - vCenter Server 5.x and earlier versions on Windows XP, 2000, and 2003: **%ALLUSERSPROFILE%\Application Data\VMware\VMware VirtualCenter\Logs**
 - vCenter Server 5.x and earlier versions on Windows Vista, 7, and 2008: **C:\ProgramData\VMware\VMware VirtualCenter\Logs**
 - vCenter Server 5.x Linux Virtual Appliance: **/var/log/vmware/vpx/**
 - vCenter Server 5.x Linux Virtual Appliance UI: **/var/log/vmware/vami/**

NOTE

If vCenter is running under a specific user account, then the logs may be located in the profile directory of that user, instead of **%ALLUSERSPROFILE%**.

5. Determine which log files you want to parse and collect.

Table 58. VMware vCenter Log Files ([Source](#))

Log File Name	Usage
vpxd.log	The main vCenter Server logs, consisting of all vSphere Client and WebServices connections, internal tasks and events, and communication with the vCenter Server Agent (vpxa) on managed ESX/ESXi hosts.
vpxd-profiler.log, profiler.log	Profiled metrics for operations performed in vCenter Server. Used by the VPX Operational Dashboard (VOD) accessible at https://VCHost/vod/index.html .
vpxd-alert.log	Non-fatal information logged about the vpxd process.
cim-diag.log and vws.log	Common Information Model monitoring information, including communication between vCenter Server and managed hosts' CIM interface
drmdump (directory)	Actions proposed and taken by VMware Distributed Resource Scheduler (DRS), grouped by the DRS-enabled cluster managed by vCenter Server. These logs are compressed.
ls.log	Health reports for the Licensing Services extension, connectivity logs to vCenter Server.
vimtool.log	Dump of string used during the installation of vCenter Server with hashed information for DNS, username and output for JDBC creation.
stats.log	Provides information about the historical performance data collection from the ESXi/ESX hosts
sms.log	Health reports for the Storage Monitoring Service extension, connectivity logs to vCenter Server, the vCenter Server database and the xDB for vCenter Inventory Service.
eam.log	Health reports for the ESX Agent Monitor extension, connectivity logs to vCenter Server.
catalina.date.log and localhost.date.log	Connectivity information and status of the VMware Webmanagement Services.
jointool.log	Health status of the VMwareVCMSDS service and individual ADAM database objects, internal tasks and events, and replication logs between linked-mode vCenter Servers.

NOTE

The various log files use different formats. You must examine your chosen file in order to determine how to parse its entries.

The main log file, **vpxd.log**, contains all login and management information. This file will be used as an example. The file has the general format of **timestamp [tag-1] [optional-tag-2] message**, and the message part might contain a multi-line trace.

vpxd.log Sample

```
2014-06-13T22:44:46.878-07:00 [04372 info 'Default' opID=DACDA564-00000004-7c] [Auth]: User
Administrator<
2014-06-13T23:15:07.222-07:00 [04136 error 'vpxdvpdxMain'] [Vpxd::ServerApp::Init] Init failed:
VpxdVdb::Init(VpxdVdb::GetVcVdbInstId(), false, false, NULL)<
--> Backtrace:<
--> backtrace[00] rip 000000018018a8ca<
--> backtrace[01] rip 0000000180102f28<
--> backtrace[02] rip 000000018010423e<
--> backtrace[03] rip 000000018008e00b<
--> backtrace[04] rip 0000000003c5c2c<
--><
```

6. Configure and restart NXLog.

Example 306. Collecting vCenter Logs Locally

In the configuration below, the `xm_multiline` extension module is used with the `HeaderLine` directive to parse log entries even when they span multiple lines. An `Exec` directive is used to drop all empty lines. A regular expression with matching groups adds fields to the event record from each log message, and the resulting log entries are sent to another host via TCP in JSON format.

nxlog.conf

```
1 <Extension vcenter>
2   Module      xm_multiline
3   HeaderLine  /(?x)(\d+-\d+-\d+T\d+:\d+:\d+)\.\d+-\d+:\d+\s+\[(.*?)\]\s+ \
4          (?:\[.*?\]\s+)?(.*)/
5   Exec        if $raw_event =~ /^$raw_event$/ drop();
6 </Extension>
7
8 <Extension _json>
9   Module      xm_json
10 </Extension>
11
12 <Input in>
13   Module      im_file
14   File        "C:\ProgramData\VMware\VMware VirtualCenter\Logs\vpwdx*.log"
15   InputType   vcenter
16   <Exec>
17     if $raw_event =~ /(?x)(\d+-\d+-\d+T\d+:\d+:\d+)\.\d+-\d+:\d+\s+\[(.*?)\]\s+ \
18          (?:\[.*?\]\s+)?((.*\s*)*)/
19     {
20       $EventTime = parsedate($1);
21       $Tag1 = $2;
22       $Tag2 = $3;
23       $Message = $4;
24     }
25   </Exec>
26 </Input>
27
28 <Output out>
29   Module      om_tcp
30   Host        192.168.1.1
31   Port        1514
32   Exec        to_json();
33 </Output>
```

Output Sample

```
{
  "EventReceivedTime": "2017-04-29 13:46:49",
  "SourceModuleName": "vcenter_in1",
  "SourceModuleType": "im_file",
  "EventTime": "2014-06-14 07:44:46",
  "Tag1": "04372 info 'Default' opID=DACDA564-00000004-7c",
  "Tag2": "",
  "Message": "[Auth]: User Administrator"
}

{
  "EventReceivedTime": "2017-04-29 13:46:49",
  "SourceModuleName": "vcenter_in1",
  "SourceModuleType": "im_file",
  "EventTime": "2014-06-14 08:15:07",
  "Tag1": "04136 error 'vpxdvpdMain'",
  "Tag2": "Vpxd::ServerApp::Init",
  "Message": "Init failed: VpxdVdb::Init(VpxdVdb::GetVcVdbInstId(), false, false, NULL)\n-->
Backtrace:\n--> backtrace[0] rip 000000018018a8ca\n--> backtrace[01] rip 0000000180102f28\n-->
backtrace[02] rip 000000018010423e\n--> backtrace[03] rip 000000018008e00b\n--> backtrace[04]
rip 0000000003c5c2c\n-->\n"
}
}
```

76.2. Remote vCenter Logging

This method of capturing vSphere logs uses a Perl script, which periodically connects to the vCenter server and retrieves logs.

TIP If you plan to use remote log collection, then we recommend setting up a dedicated user in vCenter for the log collection script to use. Using a separate user enhances security, since the log collection script connects to the vCenter web SDK remotely.

1. Download and install the latest [Perl runtime](#) and the [vSphere SDK for Perl](#).

NOTE If you are running NXLog on Windows, then the [vSphere CLI](#) is recommended instead, because it comes with the required Perl runtime environment and Vperl libraries.

2. Create a text file named `timestamp.txt`. The script will use this file to store the timestamp of the most recently downloaded log entry. The timestamp ensures that even if the vCenter server is restarted, NXLog can correctly resume log collection. Enter a timestamp in `yyyy-mm-ddThh-mm` format (for example, `2017-01-19T18:00`). Any logs with earlier timestamps will be skipped, so choose a time early enough to collect the desired logs.
3. Create a file named `getlogs.pl` with the following content.

```
getlogs.pl

#!/usr/bin/perl -w
use Encode;
use VMware::VIRuntime;
use VMware::VILib;
use Getopt::Long;
use IO::File;
use POSIX;

my $startTime;
my $stopTime;
my $server;
```

```
my $sleepTime = 60;
my $userName;
my $passWord;
my $timeStamp;
my $timeStampFile = "timestamp.txt";
my $timeNow;

my $optRs = GetOptions('s=s'=>\$server, 'u=s'=>\$userName, 'p=s'=>\$passWord, 't=s'=> \
$sleepTime, 'r=s'=>\$timeStampFile);
Getopt::Long::Configure("pass_through");

if((not defined $server) or (not defined $userName) or (not defined $passWord)){
    print "Usage: -s=server.ip.addr -u=username -p=password -t=pollinterval(seconds,
optional, default=$sleepTime) -r=timestampfile (optional, default=$timeStampFile, format:%Y-%m-
%dT%H:%M:%S)";
    exit;
}

my $file = new IO::File("< $timeStampFile");
if (defined $file) {
    my ($line);
    while (<$file>) {
        $line = $_;
        if ($line =~ /^#/) {next;} else {last;}
    }
    if (not ($line =~ /^#/)) {
        $startTime=$line;
        $startTime=~s/[\\n]+//g;
    }
    $file->close();
}

$ENV{'VI_SERVER'}=$server;
$ENV{'VI_USERNAME'}=$userName;
$ENV{'VI_PASSWORD'}=$passWord;

my %opts = (
    vmname => {
        type      => "=s",
        variable  => "name",
        help      => "Name of the Virtual Machine",
        required   => 0
    },
);

Opts::add_options(%opts);
Opts::parse();
Opts::validate();

while (1) {
    $timeNow = substr(POSIX::strftime('%Y-%m-%dT%H:%M:%S', localtime(time)), 0, 19);

    if(not defined $startTime) {
        $startTime = substr(POSIX::strftime('%Y-%m-%dT%H:%M:%S', localtime(time-
$sleepTime)), 0, 19);
    }
    $stopTime = $timeNow;

    eval{
        Util::connect()
    }
}
```

```

};

if($@){
    print
"\\"EventTime\":"\\$timeNow\", \"Message\":"\\$@\", \"UserName\":"\\$userName\"\}\n";
}
eval{
    getLog()
};
if($@){
    print
"\\"EventTime\":"\\$timeNow\", \"Message\":"\\$@\", \"UserName\":"\\$userName\"\}\n";
}
eval{
    Util::disconnect()
};
sleep($sleepTime);
}

sub getLog {
my $eventNum=0;
my ($eventArray,$lSize);
my $getFunc = Vim::get_service_content()->eventManager;
my $eventManager = Vim::get_view(mo_ref => $getFunc);
    my $eventFilter = EventFilterSpecByTime->new(beginTime=>$startTime,endTime=>$stopTime);
my $filterSpec = EventFilterSpec->new(time=>$eventFilter);
my $collectorViewRef= $eventManager->CreateCollectorForEvents(filter=>$filterSpec);
my $collectorView = Vim::get_view(mo_ref => $collectorViewRef);
my $separator="\":\"";
my $separator2="\",\"";

while (1) {
    $eventArray=$collectorView->ReadNextEvents(maxCount=>400);
    my @evtArray = @{$eventArray};
    if (not defined $eventArray) { last;}

    $lSize = @evtArray;
    if ($lSize>=1) {
        foreach (@evtArray){
            my $fullMsg = decode_utf8($_->fullFormattedMessage);
            $fullMsg=~s/[^\x{00-\x{7f}]/g;
            $fullMsg=~s/[\n]+//g;
            my $packetContent="";
            if (defined($_->createdTime)) {

                $packetContent=$packetContent."\"EventTime".$separator.$_->createdTime.$separator2.
                "Message".$separator.$fullMsg.$separator2.
                "UserName".$separator.$_->userName."\"";
                if ($startTime eq $_->createdTime) {next;}
                $timeStamp=$_->createdTime;
            }
            if (defined($_->vm)) {
                my $vmObj = $_->vm;
                $packetContent=$packetContent.",\"".VirtualMachine".$separator.$vmObj->name."\"";
            }
            if (defined($_->host)) {
                my $hostObj=$_->host;
                $packetContent=$packetContent.",\"".HostName".$separator.$hostObj->name."\"";
            }
            $packetContent.="}\n";
            print $packetContent;
        }
    }
}
}

```

```
    $eventNum++;
}
}

else {
    if (defined $timeStamp) {
        $startTime = $timeStamp;
        my $file = new IO::File("> $timestampFile");
        $file->print($timeStamp);
        $file->close();
    }
    last;
}
}
```

4. Test the script with the vCenter host. Substitute the correct IP address and credentials for the vCenter server. The **-t** is optional and sets the time between connections (the default of 60 seconds is the minimum recommended). The **-r** is also optional and can be used to specify a different location for the timestamp file. Events such as connection or authentication errors are also logged to standard output.

```
perl getlogs.pl -s=serverip -u=username -p=password -t=pollinterval -r=timestampfile
```

Output Sample

```
{
    "EventTime": "2014-06-20T18:00:00.163Z",
    "Message": "User Administrator@192.168.71.1 logged in as VI Perl",
    "UserName": "Administrator"
}
{
    "EventTime": "2014-06-20T10:56:23",
    "Message": "Error: Cannot complete login due to an incorrect user name or password.",
    "UserName": "Administrator"
}
```

5. Configure and restart NXLog.

Example 307. Collecting vCenter Logs Remotely

This configuration uses the `im_exec` module to run the Perl script and accept logs from its standard output. The `parse_json()` procedure will set fields from the JSON input in the event record for further processing.

nxlog.conf

```
1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 <Input in>
6   Module im_exec
7   # For users who have the VMware CLI installed:
8   Command "C:\Program Files (x86)\VMware\VMware vSphere CLI\Perl\bin\perl.exe"
9   # For Linux and regular Perl users this would be sufficient:
10  #Command perl
11  Arg "C:\scripts\getlogs.pl"
12  Arg -u
13  Arg <username>
14  Arg -p
15  Arg <password>
16  Arg -s
17  Arg <server_ip_addr>
18
19  # Parse JSON into fields for later processing if required
20  Exec parse_json();
21 </Input>
```

Chapter 77. Windows EventLog

The Windows EventLog captures details of both system and application events. Windows stores this data in a proprietary binary format. NXLog can be used to capture and process this EventLog data.

NXLog provides four modules for capturing EventLog data.

- The [im_msvisalog](#) module is available on Windows only, and captures EventLog data from Windows 2008/Vista and later. It can be configured to collect EventLog data from the local system or from a remote system via MSRPC (MSRPC is supported by NXLog Enterprise Edition only). See [Local Collection With im_msvisalog](#) and [Remote Collection With im_msvisalog](#).
- The [im_wseventing](#) module is available on both Linux and Windows (NXLog Enterprise Edition only). With it, EventLog data can be received from remote Windows systems using Windows Event Forwarding. This is the recommended module for most cases where remote capturing is required, because it is not necessary to specify each host that EventLog data will be captured from. See [Remote Collection With im_wseventing](#).
- The [im_wmi](#) module is available on Linux only (NXLog Enterprise Edition only). It can be used to accept EventLog data remotely over Windows Management Instrumentation (WMI). See [Remote Collection With im_wmi](#).
- The [im_mseventlog](#) module is available on Windows only, and captures EventLog data locally from Windows XP, Windows 2000, and Windows 2003. See [Local Collection With im_mseventlog](#).

The [Forwarding EventLog](#) section covers details about sending the captured EventLog data to a remote agent in BSD Syslog, JSON, or Snare formats.

For information and examples about filtering the Windows EventLog data, see the Sysmon [Advanced Filtering Options](#) section.

77.1. Local Collection With im_msvisalog

The [im_msvisalog](#) module can capture EventLog data from the local system running Windows 2008/Vista or later.

NOTE

Because the Windows EventLog subsystem does not support subscriptions to the Debug and Analytic channels, these types of events can not be collected with the [im_msvisalog](#) module.

Example 308. Collecting EventLog Locally From Windows 2008/Vista or Later

In this example, NXLog reads all events from the local Windows EventLog. The data is converted to JSON format and written to a local file.

nxlog.conf

```
1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 <Input eventlog>
6   Module im_msvisalog
7 </Input>
8
9 <Output file>
10  Module om_file
11  File   'C:\test\sysmon.json'
12  Exec   to_json();
13 </Output>
```

The [im_msvistalog](#) module can be configured with the [Query](#) directive (or the [QueryXML](#) directive) to collect only the required EventLog data.

Example 309. Querying a Subset of EventLog Data

Here, NXLog queries the local Windows EventLog for operational events only.

nxlog.conf

```
1 <Input in>
2   Module im_msvistalog
3   <QueryXML>
4     <QueryList>
5       <Query Id="0">
6         <Select Path="Microsoft-Windows-Sysmon/Operational">*</Select>
7       </Query>
8     </QueryList>
9   </QueryXML>
10 </Input>
```

This query omits security incidents below level four.

nxlog.conf

```
1 <Input in>
2   Module im_msvistalog
3   <QueryXML>
4     <QueryList>
5       <Query Id="0">
6         <Select Path='Security'>*[Security/Level=4]</Select>
7       </Query>
8     </QueryList>
9   </QueryXML>
10 </Input>
```

77.2. Remote Collection With im_msvistalog

NXLog Enterprise Edition can be configured with the [im_msvistalog](#) module for collection of events generated on remote Windows systems. In this mode, it is not necessary to run an NXLog agent on the Windows systems. Instead, MSRPC is used to receive the events.

NOTE

Because the Windows EventLog subsystem does not support subscriptions to the Debug and Analytic channels, these types of events can not be collected with the [im_msvistalog](#) module.

Example 310. Receiving EventLog Data over MSRPC

In this example configuration, the `im_msvistalog` module is used to get events from a remote server named `mywindowsbox` using MSRPC.

To replicate this example in your environment, modify the `RemoteServer`, `RemoteUser`, `RemoteDomain`, and `RemotePassword` to reflect the access credentials for the target machine.

`nxlog.conf`

```
1 <Input in>
2     Module      im_msvistalog
3     <QueryXML>
4         <QueryList>
5             <Query Id='1'>
6                 <Select Path='Application'/*></Select>
7                 <Select Path='Security'/*>*[System/Level=4]</Select>
8                 <Select Path='System'/*></Select>
9             </Query>
10            </QueryList>
11        </QueryXML>
12        RemoteServer    mywindowsbox
13        RemoteUser      Administrator
14        RemoteDomain    Workgroup
15        RemotePassword  secret
16 </Input>
```

77.3. Local Collection With `im_mseventlog`

The `im_mseventlog` module can capture EventLog data from Windows XP, Windows 2000, and Windows 2003.

The module looks up the available EventLog sources stored under the registry key `SYSTEM\CurrentControlSet\Services\Eventlog`, and polls logs from each of these or only the sources defined with the `Sources` directive of the NXLog configuration.

Example 311. Forwarding EventLog Data from Windows to a Remote Host

This example shows the most basic configuration of the `im_mseventlog` module. This configuration forwards all EventLog sources listed in the Windows registry over the network to a remote NXLog instance at the IP address `192.168.1.1`.

`nxlog.conf`

```
1 <Input eventlog>
2     Module  im_mseventlog
3 </Input>
4
5 <Output tcp>
6     Module  om_tcp
7     Host    192.168.1.1
8     Port    514
9 </Output>
```

77.4. Remote Collection With `im_wseventing`

NXLog Enterprise Edition can use the `im_wseventing` module to receive Windows EventLog data from remote machines over WEF (Windows Event Forwarding). It works on both Windows and Linux hosts.

Example 312. Receiving Windows EventLog Data using the `im_wseventing` Module

This configuration receives data from all source computers, by listening on port 5985 for connections from all sources. It also shows a configured HTTPS certificate, used to secure the transfer of EventLog data.

`nxlog.conf`

```

1 <Input in>
2   Module      im_wseventing
3   ListenAddr  0.0.0.0
4   Port        5985
5   Address     https://linux.corp.domain.com:5985/wsman
6   HTTPSCertFile %CERTDIR%/server-cert.pem
7   HTTPSCertKeyFile %CERTDIR%/server-key.pem
8   HTTPSACFile  %CERTDIR%/ca.pem
9   <QueryXML>
10    <QueryList>
11      <Query Id="0" Path="Application">
12        <Select Path="Application">*</Select>
13        <Select Path="Microsoft-Windows-Winsock-AFD/Operational">*</Select>
14        <Select Path="Microsoft-Windows-Wired-AutoConfig/Operational">
15          *
16        </Select>
17        <Select Path="Microsoft-Windows-Wordpad/Admin">*</Select>
18        <Select Path="Windows PowerShell">*</Select>
19      </Query>
20    </QueryList>
21  </QueryXML>
22 </Input>
```

A query for specific hosts can be set by adding an additional `<QueryXML>` block with a `<Computer>` tag. This tag contains a pattern that NXLog matches against the name of the connecting Windows client. Computer names not matching the pattern will use the default `QueryXML` block (containing no `<Computer>` tag). The following `QueryXML` block, if added to the above configuration, would provide an alternate query for computer names matching the pattern `foo*`.

`nxlog.conf`

```

1 <QueryXML>
2   <QueryList>
3     <Computer>foo*</Computer>
4     <Query Id="0" Path="Application">
5       <Select Path="Application">*</Select>
6     </Query>
7   </QueryList>
8 </QueryXML>
```

77.5. Remote Collection With `im_wmi`

The `im_wmi` module, available in NXLog Enterprise Edition, can be used on a system running Linux to receive EventLog data from a Windows system remotely.

NOTE	Fields from <code>EventData</code> are not transferred when using the <code>im_wmi</code> module, because Windows Management Instrumentation only provides mappings for the old-style EventLog structure (the <code>Win32_NTLogEvent</code> class).
NOTE	The <code>im_wmi</code> module offers lower performance than other approaches such as <code>im_wseventing</code> .

Example 313. Receiving EventLog Data on Linux With im_wmi

With this configuration, NXLog will connect to 10.0.2.42 and log in as Administrator to collect EventLog data.

nxlog.conf

```
1 <Input in>
2   Module    im_wmi
3   Host      10.0.2.42
4   Username  Administrator
5   Password  secret
6   Domain   WORKGROUP
7 </Input>
```

77.6. Forwarding EventLog

After collecting the EventLog data from a Windows system with NXLog, it may need to be sent to another host. This section provides details and examples for configuring this.

Event descriptions in EventLog data may contain tabs and newlines, but these are not supported by some formats like BSD Syslog. In this case, a regular expression can be used to remove them.

Example 314. Removing Tabs and Newline Sequences

This input instance is configured to modify the **\$Message** field (the event description) by replacing all tab characters and newline sequences with spaces.

nxlog.conf

```
1 <Input in>
2   Module  im_mseventlog
3   Exec    $Message =~ s/(\t|\r)/ /g;
4 </Input>
```

77.6.1. Forwarding EventLog in BSD Syslog Format

EventLog data is commonly sent in the BSD Syslog format. This can be generated with the [to_syslog_bsd\(\)](#) procedure provided by the *xm_syslog* module. For more information, see [Sending Syslog to a Remote Logger via UDP, TCP, or TLS](#).

Example 315. Sending EventLog in BSD Syslog Format

This example configuration removes tab characters and newline sequences from the **\$Message** field, converts the event record to BSD Syslog format, and forwards the event via UDP.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input eventlog>
6     Module im_msvisatalog
7     Exec   $Message =~ s/(\t|\R)/ /g; to_syslog_bsd();
8 </Input>
9
10 <Output udp>
11    Module om_udp
12    Host   10.10.1.1
13    Port   514
14 </Output>
```

NOTE

The `to_syslog_bsd()` procedure will use only a [subset](#) of the EventLog fields.

Output Sample

```
<14>Jan 2 10:21:16 win7host Service_Control_Manager[448]: The Computer Browser service entered
the running state.«
```

77.6.2. Forwarding EventLog in JSON Format

To preserve all EventLog fields, the logs can be formatted as JSON. The `xm_json` module provides a [to_json\(\)](#) procedure for this purpose. For more information about generating logs in JSON format, see [JSON](#).

Example 316. Sending EventLog in JSON Format

This example configuration converts the event record to JSON format and forwards the event via TCP.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Input eventlog>
6     Module im_msvistalog
7     Exec to_json();
8 </Input>
9
10 <Output tcp>
11    Module om_tcp
12    Host   192.168.10.1
13    Port   1514
14 </Output>
```

Output Sample

```
{
  "EventTime": "2017-01-02 10:21:16",
  "Hostname": "win7host",
  "Keywords": -9187343239835812000,
  "EventType": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "EventID": 7036,
  "SourceName": "Service Control Manager",
  "ProviderGuid": "{525908D1-A6D5-5695-8E2E-26921D2011F3}",
  "Version": 0,
  "Task": 0,
  "OpcodeValue": 0,
  "RecordNumber": 2629,
  "ProcessID": 448,
  "ThreadID": 2872,
  "Channel": "System",
  "Message": "The Computer Browser service entered the running state.",
  "param1": "Computer Browser",
  "param2": "running",
  "EventReceivedTime": "2017-01-02 10:21:17",
  "SourceModuleName": "eventlog",
  "SourceModuleType": "im_msvistalog"
}
```

For compatibility with logging systems that require BSD Syslog, the JSON format can be used with a BSD Syslog header.

Example 317. Encapsulating JSON EventLog in BSD Syslog

This example configuration converts the event record to JSON, adds a BSD Syslog header, and forwards the event via UDP.

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Extension _syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input eventlog>
10    Module im_msvistalog
11    Exec   $Message = to_json(); to_syslog_bsd();
12 </Input>
13
14 <Output udp>
15     Module om_udp
16     Host   192.168.2.1
17     Port   514
18 </Output>
```

Output Sample

```
<14>Jan  2 10:21:16 win7host Service_Control_Manager[448]: {"EventTime":"2017-01-02
10:21:16", "Hostname":"win7host", "Keywords":-
9187343239835811840, "EventType":"INFO", "SeverityValue":2, "Severity":"INFO", "EventID":7036, "Sour
ceName":"Service Control Manager", "ProviderGuid":"{525908D1-A6D5-5695-8E2E-
26921D2011F3}", "Version":0, "Task":0, "OpcodeValue":0, "RecordNumber":2629, "ProcessID":448, "Thread
ID":2872, "Channel":"System", "Message":"The Computer Browser service entered the running
state.", "param1":"Computer Browser", "param2":"running", "EventReceivedTime":"2017-01-02
10:21:17", "SourceModuleName":"eventlog", "SourceModuleType":"im_msvistalog"}<
```

77.6.3. Forwarding EventLog in the Snare Format

The Snare format is often used for Windows EventLog data. The *xm_syslog* module includes a [to_syslog_snare\(\)](#) procedure which can generate the Snare format with a Syslog header. For more information about the Snare format, see [Snare](#).

Example 318. Sending EventLog in Snare Format

This example configuration removes tab characters and newline sequences from the **\$Message** field, converts the event record to the Snare over Syslog format, and forwards the event via UDP.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input eventlog>
6     Module im_msvisalog
7     Exec   $Message =~ s/(\t|\r)/ /g; to_syslog_snare();
8 </Input>
9
10 <Output snare>
11    Module om_udp
12    Host   192.168.1.1
13    Port   514
14 </Output>
```

Output Sample

```
<14>Jan  2 10:21:16 win7host MSWinEventLog => 1 => System => 193 => Mon Jan  02 10:21:16 2017 =>
7036 => Service Control Manager => N/A => N/A => Information => win7host => N/A => => The Computer
Browser service entered the running state. => 2773<
```

Chapter 78. Windows PowerShell

PowerShell scripts can be integrated with NXLog for log processing tasks and configuration generation. See [Using PowerShell Scripts](#) below.

NXLog can also be set up to collect and parse log data from PowerShell sessions on the system, using several logging features provided by recent versions of PowerShell. See [Logging PowerShell Activity](#).

78.1. Using PowerShell Scripts

PowerShell scripts can be used with NXLog for generating, processing, and forwarding logs, as well as for generating configuration content.

78.1.1. Generating Logs

For generating logs, a PowerShell script can be used with the `im_exec` module. For another example, see [Collecting Audit Logs via SharePoint API](#).

Example 319. Using PowerShell to Generate Logs

This configuration uses the `im_exec` module to execute `powershell.exe` with the specified arguments, including the path to the script. The script creates an event and writes it to standard output in JSON format. The `xm_json parse_json()` procedure is used to parse the JSON so all the fields are available in the event record.

NOTE

This example requires PowerShell 3 or later to transport structured data in JSON format. If structured data is required with an earlier version of PowerShell, CSV format could be used instead; see the [next example](#).

nxlog.conf

```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 envvar systemroot
6 <Input powershell>
7     Module im_exec
8     Command "%systemroot%\System32\WindowsPowerShell\v1.0\powershell.exe"
9     # Use "-Version" to select a specific PowerShell version.
10    #Arg      "-Version"
11    #Arg      "3"
12    # Bypass the system execution policy for this session only.
13    Arg      "-ExecutionPolicy"
14    Arg      "Bypass"
15    # Skip loading the local PowerShell profile.
16    Arg      "-NoProfile"
17    # This specifies the path to the PowerShell script.
18    Arg      "-File"
19    Arg      "C:\ps_input.ps1"
20    # Any additional arguments are passed to the PowerShell script.
21    Arg      "arg1"
22 <Exec>
23     # Parse JSON
24     parse_json();
25
26     # Convert $EventTime field to datetime
27     $EventTime = parsedate($EventTime);
28 </Exec>
29 </Input>
```

ps_input.ps1

```
#Requires -Version 3

# Use this if you need 64-bit PowerShell (has no affect on 32-bit systems).
#if ($env:PROCESSOR_ARCHITECTURE -eq "AMD64") {
#    Write-Debug "Running 64-bit PowerShell."
#    &"$env:SYSTEMROOT\SysNative\WindowsPowerShell\v1.0\powershell.exe" ` 
#    -NonInteractive -NoProfile -ExecutionPolicy Bypass ` 
#    -File "$(($myInvocation.InvocationName))" $args
#    exit $LASTEXITCODE
#}

# Use this if you need 32-bit PowerShell.
#if ($env:PROCESSOR_ARCHITECTURE -ne "x86") {
#    Write-Debug "Running 32-bit PowerShell."
#    &"$env:SYSTEMROOT\SysWOW64\WindowsPowerShell\v1.0\powershell.exe" ` 
#    -NonInteractive -NoProfile -ExecutionPolicy Bypass ` 
#    -File "$(($myInvocation.InvocationName))" $args
#    exit $LASTEXITCODE
#}

$count = 0
$cpu = Get-WmiObject Win32_Processor | Select -First 1 -Expand datawidth
$os = [int]::Parse(((gwmi Win32_OperatingSystem).OSArchitecture -Split "-")[0])
$process = [IntPtr]::Size * 8

while($true) {
    $count += 1
    $now = [System.DateTime]::UtcNow

    # Set event fields
    $event = @{
        Arguments = [system.String]::Join(" ", $args);
        CPUBitness = $cpu;
        EventTime = $now.ToString('o');
        Message = "event$count";
        OSBitness = $os;
        ProcessBitness = $process;
    }

    # Return event as JSON
    $event | ConvertTo-Json -Compress | Write-Output

    Start-Sleep -Seconds 5
}
```

PowerShell 2 does not support JSON. Instead, events can be formatted as CSV and parsed with an [xm_csv](#) module instance.

Example 320. Using PowerShell 2 as Input

In this example, the PowerShell script generates output strings in CSV format. The `xm_csv parse_csv()` procedure is used to parse the CSV strings into fields in the event record. Note that the fields must be provided, sorted by name, in the `xm_csv Fields` directive (and corresponding types should be provided via the `FieldTypes` directive).

WARNING

For best results with structured data, use JSON with PowerShell 3 or later (see the [previous example](#)).

nxlog.conf

```

1 <Extension csv_parser>
2   Module      xm_csv
3   Fields      Arguments, EventTime, Message
4   FieldTypes  string, datetime, string
5 </Extension>
6
7 envvar systemroot
8 <Input powershell>
9   Module    im_exec
10  Command   "%systemroot%\System32\WindowsPowerShell\v1.0\powershell.exe"
11  Arg      "-Version"
12  Arg      "2"
13  Arg      "-ExecutionPolicy"
14  Arg      "Bypass"
15  Arg      "-NoProfile"
16  Arg      "-File"
17  Arg      "C:\ps2_input.ps1"
18  Exec     csv_parser->parse_csv();
19 </Input>
```

ps2_input.ps1

```

#Requires -Version 2

$count = 0

while($true) {
    $count += 1
    $now = [System.DateTime]::UtcNow

    # Set event fields
    $event = @{
        Arguments = [system.String]::Join(" ", $args);
        EventTime = $now.ToString('o');
        Message = "event$count";
    }

    # Return event as CSV
    $row = New-Object PSObject
    $event.GetEnumerator() | Sort-Object -Property Name | ForEach-Object {
        $row | Add-Member -MemberType NoteProperty -Name $_.Name -Value $_.Value
    }
    $row | ConvertTo-Csv -NoTypeInformation | Select-Object -Skip 1 | Write-Output

    Start-Sleep -Seconds 5
}
```

78.1.2. Forwarding Logs

For forwarding logs, a PowerShell script can be used with the `om_exec` module.

Example 321. Using PowerShell to Forward Logs

This configuration uses `om_exec` to execute `powershell.exe` with the specified arguments, including the path to the script. The script reads events on standard input.

NOTE

This configuration requires PowerShell 3 or later for its JSON support and to correctly read lines from standard input.

TIP

See the [Using PowerShell to Generate Logs](#) example above for more details about `powershell.exe` arguments and PowerShell code for explicitly specifying a 32-bit or 64-bit environment.

nxlog.conf

```

1 <Extension _json>
2   Module xm_json
3 </Extension>
4
5 envvar systemroot
6 <Output powershell>
7   Module om_exec
8   Command "%systemroot%\System32\WindowsPowerShell\v1.0\powershell.exe"
9   Arg   "-ExecutionPolicy"
10  Arg   "Bypass"
11  Arg   "-NoProfile"
12  Arg   "-File"
13  Arg   "C:\ps_output.ps1"
14  Exec  to_json();
15 </Output>
```

ps_output.ps1

```

#Requires -Version 3

while($line = [Console]::In.ReadLine()) {
  # Convert from JSON
  $record = $line | ConvertFrom-Json

  # Write out to file
  $record | Out-File -FilePath 'C:\out.log' -Append
}
```

78.1.3. Generating Configuration

A PowerShell script can be used with the `include_stdout` directive to generate dynamic NXLog configuration. NXLog will execute the script during parsing of the configuration file.

Because `include_stdout` does not support arguments, it is simplest to use a batch/PowerShell polyglot script for this purpose. For another example, see [Automatic Retrieval of IIS Site Log Locations](#).

TIP

The Command Prompt may print '`\u202a is not recognized`' if a Unicode byte order mark (BOM) is included in the batch file. To fix this, use Notepad and select the **ANSI** encoding when saving the file.

Example 322. Using PowerShell and include_stdout

This configuration uses PowerShell code to generate the [File](#) directive for the Input instance.

nxlog.conf

```
1 <Input in>
2     Module      im_file
3     include_stdout  C:\include.cmd
4 </Input>
```

include.cmd

```
@( Set "=_= (
REM " ) <#
)
@Echo Off
SetLocal EnableExtensions DisableDelayedExpansion
set powershell=powershell.exe

REM Use this if you need 64-bit PowerShell (has no affect on 32-bit systems).
REM if defined PROCESSOR_ARCHITEW6432 (
REM set powershell=%SystemRoot%\SysNative\WindowsPowerShell\v1.0\powershell.exe
REM )

REM Use this if you need 32-bit PowerShell.
REM if NOT %PROCESSOR_ARCHITECTURE% == x86 (
REM set powershell=%SystemRoot%\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
REM )

%powershell% -ExecutionPolicy Bypass -NoProfile ^
-Command "iex ((gc '%~f0') -join [char]10)"
EndLocal & Exit /B %ErrorLevel%
#>

# PowerShell code starts here.

Write-Output "File 'C:\in.log'"
```

78.2. Logging PowerShell Activity

Recent versions of Windows PowerShell provide several features for logging of activity from PowerShell sessions. NXLog can be configured to collect and parse these logs.

In addition to the sections below, see [Securing PowerShell in the Enterprise](#), [Greater Visibility Through PowerShell Logging](#), and [PowerShell ❤ the Blue Team](#). Also see the [Command line process auditing](#) article on Microsoft Docs and the [Sysmon](#) chapter, both of which can be used to generate events for command line process creation (but not for commands executed through the PowerShell engine).

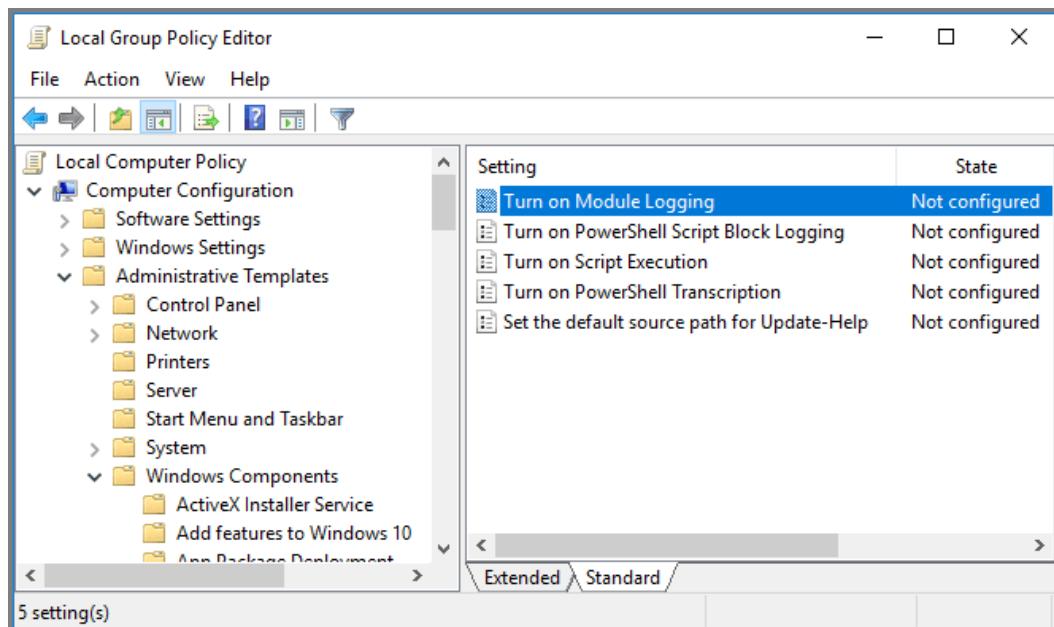
78.2.1. Module Logging

Module logging, available since PowerShell 3, logs pipeline execution events for specified PowerShell modules. This feature writes Event ID 4103 events to the Microsoft-Windows-PowerShell/Operational channel.

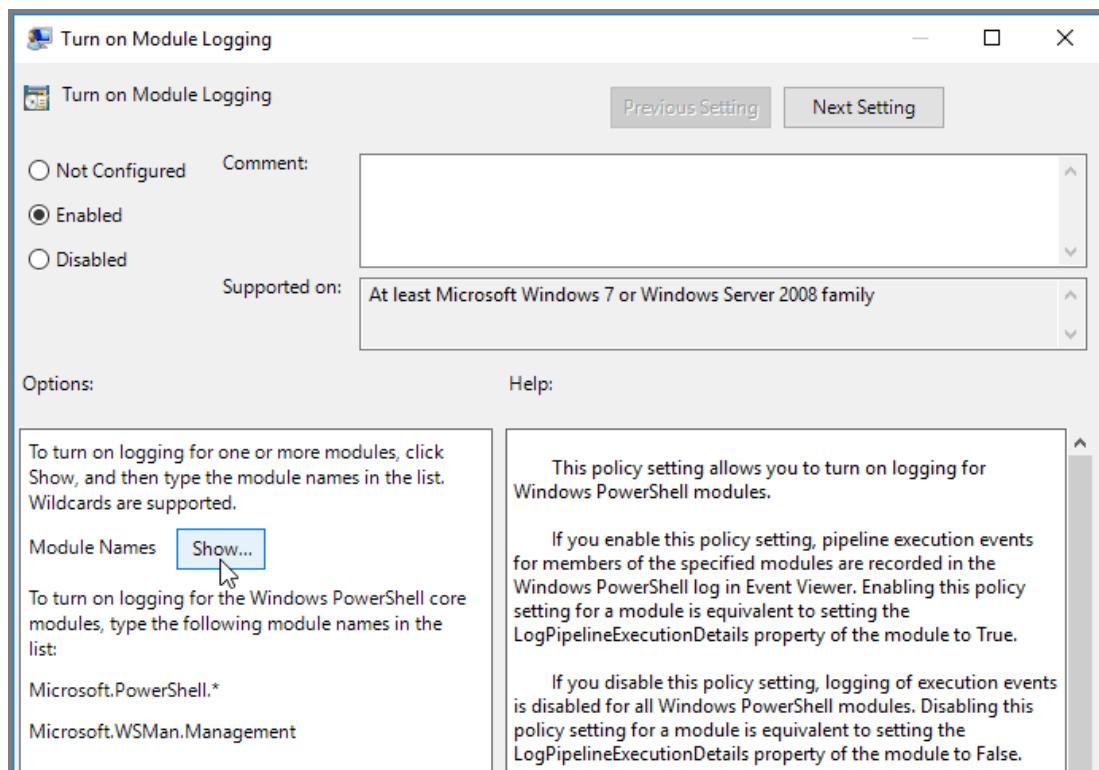
Module logging can be enabled by setting the LogPipelineExecutionDetails property of a module to True. Or this property can be enabled for selected modules through Group Policy as follows.

1. Open the Group Policy MMC snapin ([gpedit.msc](#)).

2. Go to **Computer Configuration > Administrative Templates > Windows Components > Windows PowerShell** and open the **Turn on Module Logging** setting.



3. Select **Enabled**. Then click the **[Show...]** button and enter the modules for which to enable logging. Use an asterisk (*) to enable logging for all modules.



Example 323. Collecting Module Logging Events

This configuration collects all events with ID 4103 from the Windows PowerShell Operational channel. The logs are then converted to Syslog-encapsulated JSON and forwarded via TCP.

nxlog.conf

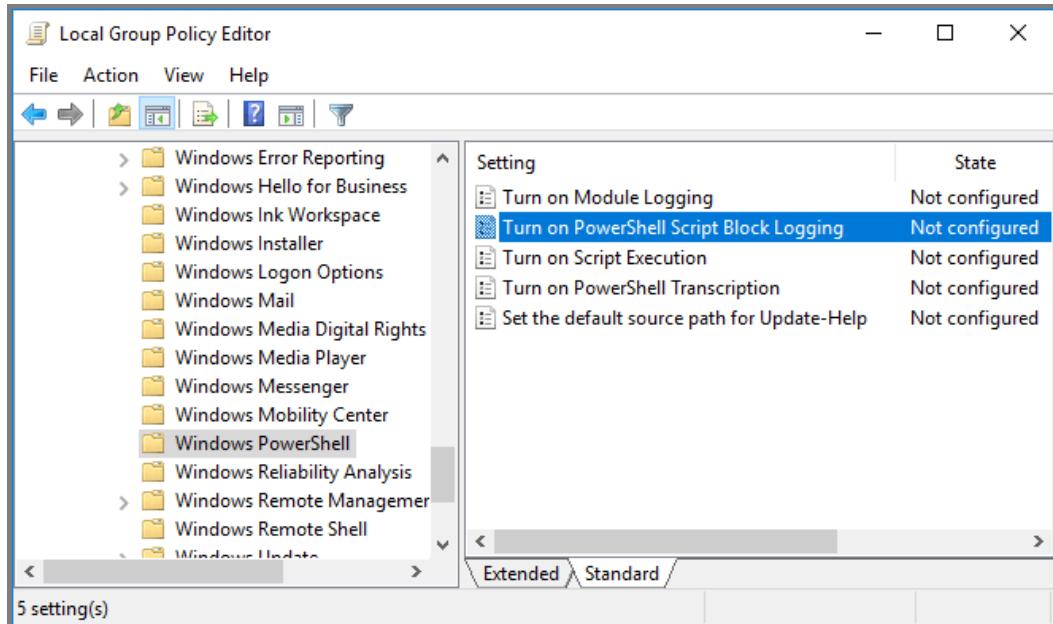
```
1 <Extension _json>
2     Module xm_json
3 </Extension>
4
5 <Extension _syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input module_logging>
10    Module im_msvisalog
11        <QueryXML>
12            <QueryList>
13                <Query Id="0" Path="Microsoft-Windows-PowerShell/Operational">
14                    <Select Path="Microsoft-Windows-PowerShell/Operational">
15                        *[System[EventID=4103]]</Select>
16                </Query>
17            </QueryList>
18        </QueryXML>
19        Exec $Message = to_json(); to_syslog_bsd();
20 </Input>
21
22 <Output tcp>
23     Module om_tcp
24     Host   10.12.1.1
25     Port   514
26 </Output>
```

78.2.2. Script Block Logging

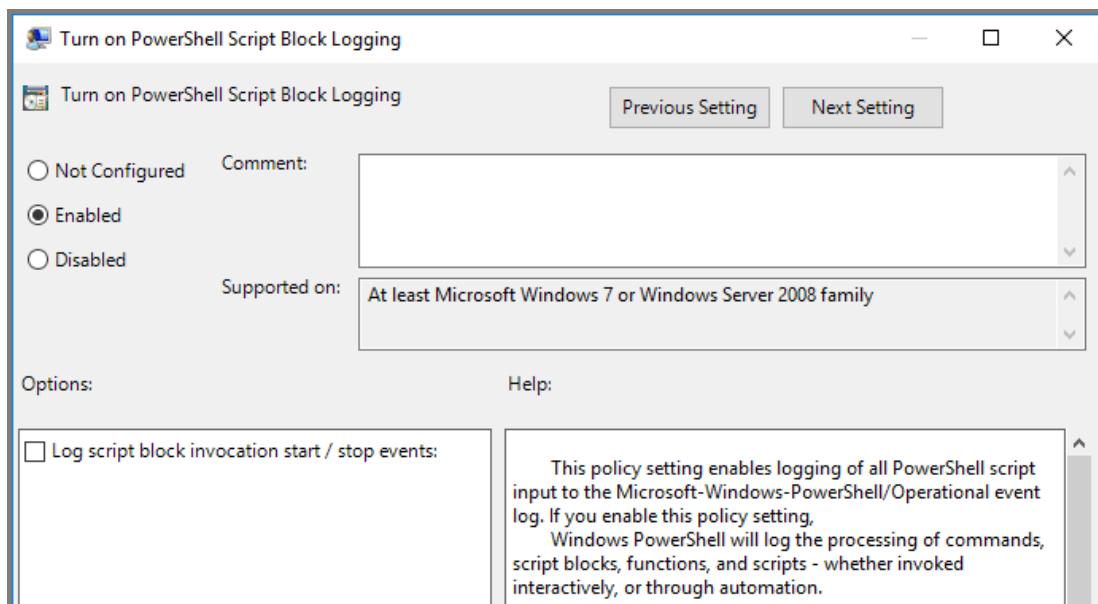
PowerShell 5 introduces script block logging, which records the content of all script blocks that are processed. Events with ID 4104 are written to the Microsoft-Windows-PowerShell/Operational channel. Start and stop events can also be enabled; these events have IDs 4105 and 4106.

Script block logging can be configured through Group Policy as follows.

1. Open the Group Policy MMC snapin ([gpedit.msc](#)).
2. Go to **Computer Configuration** > **Administrative Templates** > **Windows Components** > **Windows PowerShell** and open the **Turn on PowerShell Script Block Logging** setting.



3. Select **Enabled**. Optionally, check the **Log script block invocation start/stop events** option (this will generate a high volume of event logs).



Example 324. Collecting Script Block Logging Events

The following configuration collects events with IDs 4104, 4105, and 4106 from the Windows PowerShell Operational channel. Verbose level events are excluded.

nxlog.conf

```
1 <Input script_block_logging>
2     Module im_msvisatalog
3     <QueryXML>
4         <QueryList>
5             <Query Id="0" Path="Microsoft-Windows-PowerShell/Operational">
6                 <Select Path="Microsoft-Windows-PowerShell/Operational">
7                     *[System[(Level=0 or Level=1 or Level=2 or Level=3 or Level=4)
8                         and ((EventID >= 4104 and EventID <= 4106))]]
9                 </Select>
10            </Query>
11        </QueryList>
12    </QueryXML>
13 </Input>
```

78.2.3. Transcription

PowerShell provides "over-the-shoulder" transcription of PowerShell sessions with the `Start-Transcript` cmdlet. With PowerShell 5, system-wide transcription can be enabled via Group Policy; this is equivalent to calling the `Start-Transcript` cmdlet on each PowerShell session. Transcriptions are written to the current user's Documents directory unless a system-level output directory is set in the policy settings.

Log Sample (With Invocation Headers Enabled)

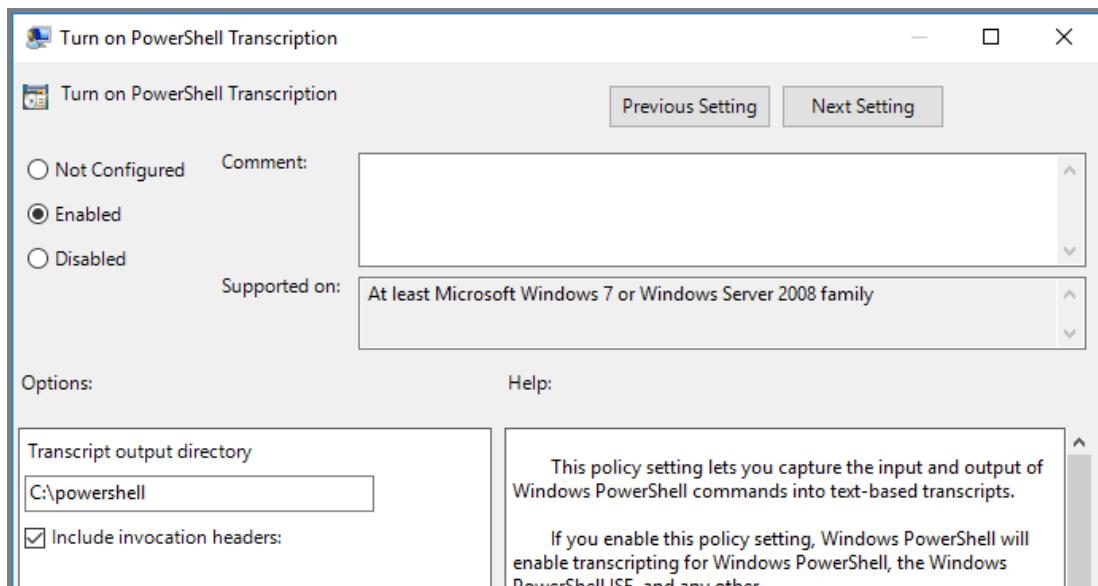
```
*****
Windows PowerShell transcript start
Start time: 20171030223248
Username: WIN-FT17VBNL4B2\Administrator
RunAs User: WIN-FT17VBNL4B2\Administrator
Machine: WIN-FT17VBNL4B2 (Microsoft Windows NT 10.0.14393.0)
Host Application: C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell.exe
Process ID: 4268
PSVersion: 5.1.14393.1770
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.14393.1770
BuildVersion: 10.0.14393.1770
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
*****
*****
Command start time: 20171030223255
*****
PS C:\Users\Administrator> echo test
test
*****
Command start time: 20171030223256
*****
PS C:\Users\Administrator> exit
*****
Windows PowerShell transcript end
End time: 20171030223256
*****
```

System-wide transcription can be enabled through Group Policy as follows.

WARNING

If system-wide transcription to a shared location is enabled, access to that directory should be limited to prevent users from viewing the transcripts of other users or computers.

1. Open the Group Policy MMC snapin ([gpedit.msc](#)).
2. Go to **Computer Configuration > Administrative Templates > Windows Components > Windows PowerShell** and open the **Turn on PowerShell Transcription** setting.
3. Select **Enabled**. Set a system-wide transcript output directory if required. Check the **Include invocation headers** option (this setting generates a timestamp for each command and is recommended).



Example 325. Parsing PowerShell Transcriptions

This configuration reads and parses transcript files written to the **TRANSCRIPTS_DIR** directory (which should be set appropriately). Headers, footers, and commands are parsed as separate events. **\$File** and **\$EventTime** fields are set for each event (invocation headers must be enabled for command timestamps). **\$Command** and **\$Output** fields are added for command events. Fields from the header entries are parsed with **xm_kvp** and added to the event record. Finally, the logs are converted to JSON format and forwarded via TCP.

NOTE

The **HeaderLine** below must be changed if invocation headers are not enabled. See the comment in the configuration.

nxlog.conf

```

1 define TRANSCRIPTS_DIR C:\powershell
2
3 <Extension _json>
4   Module xm_json
5 </Extension>
6
7 <Extension transcript_parser>
8   Module xm_multiline
9   # Use this if invocation headers are ON (recommended)
10  HeaderLine /^.*{22}$/
11  # Use this if invocation headers are OFF (not recommended)
12  #HeaderLine /^(\*{22}PS[^>]|PS[^>]\*{22})/
13  <Exec>
```

```
14     $raw_event =~ s/^xEF\xBB\xBF//;
15     if get_var('include_next_record') and $raw_event =~ /^*\{22\}$/
16     {
17         set_var('include_next_record', FALSE);
18         $raw_event =~ s/^*\*/;
19     }
20     else if $raw_event =~ /Command start time: \d{14}/
21     {
22         set_var('include_next_record', TRUE);
23     }
24 </Exec>
25 </Extension>
26
27 <Extension transcript_header_parser>
28     Module      xm_kvp
29     KVPDelimiter \n
30 </Extension>
31
32 <Input transcription>
33     Module      im_file
34     File        '%TRANSCRIPTS_DIR%\PowerShell_transcript.*'
35     InputType   transcript_parser
36     <Exec>
37         # Add file name as field to event record
38         $File = file_name();
39
40         # Drop extra separator line events
41         if $raw_event =~ /^*\{22\}$/
42             drop();
43
44         # Parse header and footer entries
45         else if $raw_event =~ /(x)^*\{22\}\r\n
46             (?<Message>Windows\ PowerShell\ transcript\ (start|end)\r\n
47             (?<KVP>.*))$/s
48         {
49             $KVP =~ s/\r//g;
50             transcript_header_parser->parse_kvp($KVP);
51             delete($KVP);
52             if ${Start time}
53                 $EventTime = strftime(${Start time}, '%Y%m%d%H%M%S');
54             else
55                 $EventTime = strftime(${End time}, '%Y%m%d%H%M%S');
56         }
57
58         # Parse command entries with invocation headers
59         else if $raw_event =~ /(x)^*\{22\}\r\n
60             (?<Part1>Command\ start\ time:\ (?<EventTime>\d{14})\r\n
61             (?<Part2>\*{21}\r\nPS[^>]*>\ ?
62                 (?<Command>[\r\n]+)(\r\n(?<Output>.+))?)$/
63         {
64             $Message = $Part1 + '*' + $Part2;
65             delete($Part1);
66             delete($Part2);
67             $EventTime = strftime($EventTime, '%Y%m%d%H%M%S');
68         }
69
70         # Parse command entries without invocation headers
71         else if $raw_event =~ /(x)^PS[^>]*>\ ?
72             (?<Command>[\r\n]+)(\r\n(?<Output>.+))?$/
73         {
```

```

74             $Message = $raw_event;
75         }
76     else log_warning('Failed to parse transcript entry');
77 </Exec>
78 </Input>
79
80 <Output out>
81     Module  om_tcp
82     Host    10.12.1.1
83     Port    514
84     Exec    to_json();
85 </Output>

```

The following output shows the first two events of the [log sample](#) above.

Output Sample

```
{
  "EventReceivedTime": "2017-10-30 22:32:49",
  "SourceModuleName": "transcription",
  "SourceModuleType": "im_file",
  "File": "C:\\powershell\\\\\\20171030\\\\PowerShell_transcript.WIN-
FT17VBNL4B2.LcxuCZbr.20171030223248.txt",
  "Message": "Windows PowerShell transcript start\r\nStart time: 20171030223248\r\nUsername:
WIN-FT17VBNL4B2\\Administrator\r\nRunAs User: WIN-FT17VBNL4B2\\Administrator\r\nMachine: WIN-
FT17VBNL4B2 (Microsoft Windows NT 10.0.14393.0)\r\nHost Application: C:\\Windows\\system32
\\WindowsPowerShell\\v1.0\\PowerShell.exe\r\nProcess ID: 4268\r\nPSVersion: 5.1.14393.1770
\r\nPSEdition: Desktop\r\nPSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.14393.1770
\r\nBuildVersion: 10.0.14393.1770\r\nCLRVersion: 4.0.30319.42000\r\nWSManStackVersion: 3.0
\r\nPSRemotingProtocolVersion: 2.3\r\nSerializationVersion: 1.1.0.1",
  "StartTime": "20171030223248",
  "Username": "WIN-FT17VBNL4B2\\Administrator",
  "RunAsUser": "WIN-FT17VBNL4B2\\Administrator",
  "Machine": "WIN-FT17VBNL4B2 (Microsoft Windows NT 10.0.14393.0)",
  "HostApplication": "C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\PowerShell.exe",
  "ProcessID": "4268",
  "PSVersion": "5.1.14393.1770",
  "PSEdition": "Desktop",
  "PSCompatibleVersions": "1.0, 2.0, 3.0, 4.0, 5.0, 5.1.14393.1770",
  "BuildVersion": "10.0.14393.1770",
  "CLRVersion": "4.0.30319.42000",
  "WSManStackVersion": "3.0",
  "PSRemotingProtocolVersion": "2.3",
  "SerializationVersion": "1.1.0.1",
  "EventTime": "2017-10-30 22:32:48"
}
{
  "EventReceivedTime": "2017-10-30 22:32:56",
  "SourceModuleName": "transcription",
  "SourceModuleType": "im_file",
  "File": "C:\\powershell\\\\\\20171030\\\\PowerShell_transcript.WIN-
FT17VBNL4B2.LcxuCZbr.20171030223248.txt",
  "Command": "echo test",
  "EventTime": "2017-10-30 22:32:55",
  "Output": "test",
  "Message": "Command start time: 20171030223255\r\n*****\r\nPS C:\\\\Users
\\\\Administrator> echo test\r\\ntest"
}
```

Troubleshooting

Chapter 79. NXLog's Internal Logs

While NXLog is a tool for processing log data from other systems, it will also generate log messages about its own operations. These messages are essential for troubleshooting problems, and should be checked first if NXLog is not functioning as expected.

Internal logs are written to the file set by the [LogFile](#) directive. If this directive is not specified in the configuration, add it to enable logging to file.

NOTE

Some Windows applications (WordPad, for example) cannot open the log file while the NXLog process is running because of exclusive file locking. Use a viewer that does not lock the file, like Notepad.

79.1. Log Level

By default, internal logs are generated with a log level of INFO. To get more detailed information about NXLog's operations, set the [LogLevel](#) directive to DEBUG. Because this can produce an extreme amount of logs, you should only enable this for troubleshooting.

79.2. Injecting Internal Logs into a Route

NXLog can also write internal log data into a normal route using the [im_internal](#) module. Internal log messages can then be forwarded like any other log source.

WARNING

Local logging is more fault-tolerant than routed logging, and is therefore recommended for troubleshooting.

NOTE

It is not possible to use a log level higher than INFO with the [im_internal](#) module. DEBUG level messages can only be written to the local log file.

79.3. Running NXLog in the Foreground

When running in the foreground, NXLog prints internal logs to the standard output and standard error streams, which are both visible in the terminal.

Use `nxlog -f` to run NXLog in the foreground.

79.4. Generating Additional Log Messages in your Configuration

It may be helpful to add extra logging statements to your configuration using the [log_info\(\)](#) procedure. Anything that is printed with `log_info()` or related procedures will appear in the configured log file, on the standard output and standard error streams, and in the [im_internal](#) log source (DEBUG level messages are not available via the `im_internal` module).

Example 326. Using the log_info() Procedure

In this example, log messages are accepted over UDP on port 514. If "keyword" is found in the unparsed message, an INFO level message will be generated.

nxlog.conf

```
1 <Input in>
2     Module im_udp
3     Port   514
4     <Exec>
5         if $raw_event =~ /keyword/
6             log_info("FOUND KEYWORD IN MSG: [" + $raw_event + "]");
7     </Exec>
8 </Input>
```

Chapter 80. Resolving Common Issues

80.1. "Nxlog failed to start"

You may receive this error message in the log file when NXLog fails to start (line break added):

```
nxlog failed to start: Invalid keyword: ýþ# at \r\n
C:\Program Files (x86)\nxlog\conf\nxlog.conf:1\r\n
```

This issue occurs because the NXLog configuration file has been saved in either UTF-16 text encoding, or UTF-8 text encoding with a BOM header.

Open the configuration file in a text editor and save it using ASCII encoding or plain UTF-8.

TIP On Windows, you can use Notepad to correct the text encoding of this file.

80.2. "Permission denied"

When configured to read from a file in the /var/log directory, NXLog may log the following error:

```
ERROR failed to open /var/log/messages;Permission denied\r\n
```

This error occurs because NXLog does not have permission to read the file with the configured **User** and **Group**. See [Reading Rsyslog Log Files](#) for more information about using NXLog to read from files in /var/log.

80.3. Log Entries are Concatenated with Logstash

If you are using Logstash and find that log entries are concatenated, make sure that you are using the **json_lines** codec in your Logstash server configuration.

The default **json** codec in Logstash sometimes fails to parse log entries passed from NXLog. Switch to the **json_lines** codec for better reliability.

80.4. Log File is in Use by Another Application

When trying to view the NXLog's internal log file on Windows, you may receive an error message indicating that the log file is in use by another application and cannot be accessed.

To resolve this issue, either:

- open the log file with an application that does not use exclusive locking (such as Notepad), or
- stop NXLog before opening the log file.

80.5. "Connection refused" with im_tcp or im_ssl

When using the **im_tcp** and **im_ssl** modules to transfer data over the network, firewalls and other network issues can prevent successful connections. This can result in **Connection refused** errors.

To resolve this issue:

1. check that no firewall, gateway, or other network issue is blocking the connection; and
2. verify that the system can resolve the host name used in the **Host** directive of the configuration file.

80.6. "Missing logdata"

Sometimes NXLog will try to evaluate a directive, but the required log data is not available in the current context. This will cause any dependent operations to fail, the directive to terminate, and the following error to be logged:

```
missing logdata, assignment possibly after drop()
```

This error will occur when attempting to access a field, either:

- after running the [drop\(\)](#) procedure, or
- from the [Exec](#) directive of a Schedule block.

In both cases, the log data is not available in the current context. Data from a dropped event is no longer available, and log data will never be available to a scheduled [Exec](#) directive, because its execution is not triggered by a log message.

An attempt to access a field can occur directly (with a field assignment) or indirectly (by calling a function or procedure that accesses log data).

Example 327. Assignment After drop()

With this configuration, NXLog drops the message if the `$raw_event` field matches the keyword. However, the next statement tries to access a field from the message, even though it may have been dropped.

nxlog.conf (Incorrect)

```
1 <Input in>
2   Module  im_udp
3   Port    514
4   <Exec>
5     if $raw_event =~ /keyword/
6     {
7       drop();
8       $EventTime = now();
9     }
10  </Exec>
11 </Input>
```

If the message is not found, then the second statement fails and NXLog generates the following error:

```
missing logdata, assignment possibly after drop()
```

To resolve this issue, use a conditional statement to only reference fields that are available. Or use the [dropped\(\)](#) function, which will return TRUE if the current event has already been dropped.

Example 328. Correctly Using drop()

This configuration only attempts to do the field assignment if the regular expression does not match.

nxlog.conf

```
1 <Input in>
2   Module im_udp
3   Port 514
4   <Exec>
5     if $raw_event =~ /keyword/
6       drop();
7     else
8       $EventTime = now();
9   </Exec>
10 </Input>
```

80.7. Windows EventLog Error: "ignoring source"

When NXLog is set up to access the Windows EventLog, the permissions may not be sufficient. In this case, the NXLog log files show errors such as:

```
2013-01-10 13:43:30 WARNING ignoring source as it cannot be subscribed to (error code: 5)←
```

If this occurs, use the [wevutil](#) utility to grant the new user access to the Windows EventLog. See this TechNet article for more details about the procedure: [Giving Non Administrators permission to read Event Logs Windows 2003 and Windows 2008](#).

80.8. Processing Stops if an Output Fails

NXLog can send one log stream to multiple outputs. This can be configured by either using the same input in multiple routes or using multiple outputs in the same route (see [Routes](#)). By default, when one of the outputs fails NXLog will stop sending logs to all outputs. This is caused by NXLog's [flow control](#) mechanism, which is designed to prevent messages from being lost. Flow control pauses an Input or Processor module when the next module in the route is not ready to accept data.

In some cases, it is preferred for NXLog to continue sending logs to the remaining active output and discard logs for the failed output. The simplest solution is to disable flow control. This can be done globally with the global [FlowControl](#) directive, or for the corresponding Input (and Processor, if any) modules only, with the module [FlowControl](#) directive. Note that with flow control disabled, an Input or Processor module will continue to process logs even if the next module's buffers are full (and the logs will be dropped).

To retain the improved message durability provided by flow control, it is possible to instead explicitly specify when to drop logs by using a separate route for each output that may fail. Add a [pm_buffer](#) module instance to that route, and configure the buffer to drop logs when it reaches a certain size. The output that follows the buffer can fail without causing any Input or Processor module before the buffer to pause processing. See [Explicit Drop](#) for more information.

Chapter 81. Debugging Data Processing

This section explains how to resolve issues with complex processing rules and configurations while using NXLog.

The examples in this section show how the event log stream can be debugged. Using these methods, you can see:

- the data that has been received or read by the inputs;
- whether a required field exists, and what its value is;
- whether the parser is working correctly, and whether it is populating the fields correctly; and
- the contents of all fields, and their values after NXLog parses them.

NOTE

A remote device may be sending invalid data to NXLog. To troubleshoot in this case, use a network traffic analyzer such as Wireshark or tcpdump.

Example 329. Writing Fields and Values to an External File

In this example configuration, the `file_write()` procedure (from the `xm_fileop` module) is used to dump information to an external file.

nxlog.conf

```
1 <Extension _fileop>
2   Module xm_fileop
3 </Extension>
4
5 <Extension _syslog>
6   Module xm_syslog
7 </Extension>
8
9 <Input in>
10  Module im_tcp
11  Host 0.0.0.0
12  Port 1514
13  <Exec>
14    parse_syslog_bsd();
15
16    # Debug SyslogSeverity and Hostname fields
17    file_write("/tmp/debug.txt",
18              "Severity: " + $SyslogSeverity + ", Hostname: " + $Hostname);
19  </Exec>
20 </Input>
```

Example 330. Writing Specific Fields and Values to the Internal Log

This configuration uses the `log_info()` procedure to send values to the internal log.

The generated messages will be visible:

- in the file defined in the `LogFile` global directive,
- in the input from the `im_internal` module, and
- on standard output when running NXLog in the foreground with the `-f` command line switch.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input in>
6     Module im_tcp
7     Host   0.0.0.0
8     Port   1514
9     <Exec>
10        parse_syslog_bsd();
11
12        # Debug SyslogSeverity and Hostname fields
13        log_info("Severity: " + $SyslogSeverity + ", Hostname: " + $Hostname);
14    </Exec>
15 </Input>
```

Example 331. Dumping All Fields to the Internal Log

In this example, the `to_json()` procedure (from the `xm_json` module) is used to dump all the fields to the internal log.

NOTE

Alternatively, the `to_xml()` procedure (from the `xm_xml` module) could be used.

nxlog.conf

```
1 <Extension _syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension _json>
6     Module xm_json
7 </Extension>
8
9 <Input in>
10    Module im_tcp
11    Host   0.0.0.0
12    Port   1514
13    <Exec>
14        parse_syslog_bsd();
15
16        # Dump $raw_event
17        log_info("raw_event is: " + $raw_event);
18
19        # Dump fields in JSON
20        log_info("Other fields are: " + to_json());
21    </Exec>
22 </Input>
```

Output Sample

```
2012-05-18 13:11:35 INFO raw_event is: <27>2010-10-12 12:49:06 host app[12345]: test message<
2012-05-18 13:11:35 INFO Other fields are:
{"MessageSourceAddress":"127.0.0.1", "EventReceivedTime":"2012-05-18
13:11:35", "SourceModuleName":"in", "SourceModuleType":"im_tcp", "SyslogFacilityValue":3, "SyslogFa
cility":"DAEMON", "SyslogSeverityValue":3, "SyslogSeverity":"ERR", "SeverityValue":4, "Severity":"E
RROR", "Hostname":"host", "EventTime":"2010-10-12
12:49:06", "SourceName":"app", "ProcessID":"12345", "Message":"test message"}<
```

Chapter 82. Debugging NXLog

82.1. Memory Leaks

If the memory usage climbs and exceed 100 MB, there is most likely a memory leak. We recommend using Valgrind on GNU/Linux to debug memory leaks.

1. Install the debug symbols (`-dbg`) package (for example, `nxlog-dbg_3.0.1759_amd64.deb`).
2. Install Valgrind.
3. Add the `NoFreeOnExit` directive to the NXLog configuration file. This directive ensures that modules are not unloaded when NXLog is stopped so Valgrind can properly resolve backtraces into modules.

`NoFreeOnExit TRUE`

4. Start NXLog under Valgrind with the following command. If you have `User` set to `nxlog` in your `nxlog.conf` file, then the command must be executed with `su`, otherwise Valgrind will not be able to create the `massif.out` file at the end of the sampling process.

```
# cd /tmp  
# su nxlog - -c "valgrind --tool=massif --pages-as-heap=yes /usr/bin/nxlog -f"
```

5. Let NXLog run for a while until the Valgrind process shows the memory increase, then interrupt it with `Ctrl +c`. There should now be a `massif.out.xxxx` file in `/tmp`.
6. Send the `massif.out.xxxx` file with a bug report.
7. Optionally, create a report from this with the `ms_print` command:

```
$ ms_print massif.out.xxxx
```

The output of the `ms_print` report contains an ASCII chart at the top showing the increase in memory usage. The chart will show the sample number with the highest memory usage marked with `(peak)`. This will normally be at the end of the chart (the last sample). The backtrace from this sample should indicate where the most memory is allocated.

Enterprise Edition Reference Manual

Chapter 83. Man Pages

83.1. nxlog(8)

NAME

nxlog - collects, processes, converts, and forwards event logs in many different formats

SYNOPSIS

nxlog [-c *conffile*] [-f]

nxlog [-c *conffile*] -v

nxlog [-r | -s]

DESCRIPTION

NXLog can process high volumes of event logs from many different sources. Supported types of log processing include rewriting, correlating, alerting, filtering, and pattern matching. Additional features include scheduling, log file rotation, buffering, and prioritized processing. After processing, NXLog can store or forward event logs in any of many supported formats. Inputs, outputs, and processing are implemented with a modular architecture and a powerful configuration language.

While the details provided here apply to NXLog installations on Linux and other UNIX-style operating systems in particular, a few Windows-specific notes are included.

OPTIONS

-c *conffile*, **--conf** *conffile*

Specify an alternate configuration file *conffile*. On Windows, this option must be used with **-f**. To change the configuration file used by the NXLog service on Windows, modify the service parameters.

-f, **--foreground**

Run in foreground, do not daemonize.

-h, **--help**

Print help.

-r, **--reload**

Reload configuration of a running instance.

-s, **--stop**

Send stop signal to a running instance.

-v, **--verify**

Verify configuration file syntax.

SIGNALS

Various signals can be used to control the NXLog process. Some corresponding Windows control codes are also available; these are shown in parentheses where applicable.

SIGHUP

This signal causes NXLog to reload the configuration and restart the modules. On Windows, "sc stop nxlog"

and "sc start nxlog" can be used instead.

SIGUSR1 (200)

This signal generates an internal log message with information about the current state of NXLog and its configured module instances. The message will be generated with INFO log level, written to the log file (if configured with [LogFile](#)), and available via the [im_internal](#) module.

SIGUSR2 (201)

This signal causes NXLog to switch to the DEBUG log level. This is equivalent to setting the [LogLevel](#) directive to [DEBUG](#) but does not require NXLog to be restarted.

SIGINT/SIGQUIT/SIGTERM

NXLog will exit if it receives one of these signals. On Windows, "sc stop nxlog" can be used instead.

On Linux/UNIX, a signal can be sent with the [kill](#) command. The following, for example, sends the SIGUSR1 signal:

```
kill -SIGUSR1 $(cat /opt/nxlog/var/run/nxlog/nxlog.pid)
```

On Windows, a signal can be sent with the [sc](#) command. The following, for example, sends the 200 signal:

```
sc control nxlog 200
```

FILES

/opt/nxlog/bin/nxlog

The main NXLog executable

/opt/nxlog/bin/nxlog-stmmt-verifier

This tool can be used to check NXLog Language statements. All statements are read from standard input and then validated. If a statement is invalid, the tool prints an error to standard error and exits non-zero.

/opt/nxlog/etc/nxlog.conf

The default configuration file

/opt/nxlog/lib/nxlog/modules

The NXLog modules are located in this directory, by default. See the [ModuleDir](#) directive.

/opt/nxlog/spool/nxlog

If [PersistLogqueue](#) is set to TRUE, module queues are stored in this directory. See also [LogqueueDir](#) and [SyncLogqueue](#).

/opt/nxlog/spool/nxlog/configcache.dat

This is the position cache file where positions are saved. See the [NoCache](#) directive, in addition to [CacheDir](#), [CacheFlushInterval](#), and [CacheSync](#).

/opt/nxsec/var/run/nxlog/nxlog.pid

The process ID (PID) of the currently running NXLog process is written to this file. See the [PidFile](#) directive.

ENVIRONMENT

To access environment variables in the NXLog configuration, use the [envvar](#) directive.

SEE ALSO

[nxlog-processor\(8\)](#)

NXLog website: <https://nxlog.co>

NXLog User Guide: <https://nxlog.co/documentation/nxlog-user-guide>

COPYRIGHT

Copyright © NXLog Ltd. 2018

The NXLog Community Edition is licensed under the NXLog Public License. The NXLog Enterprise Edition is not free and has a commercial license.

83.2. nxlog-processor(8)

NAME

nxlog-processor - performs batch log processing

SYNOPSIS

nxlog-processor [-c *conffile*] [-v]

DESCRIPTION

The nxlog-processor tool is similar to the NXLog daemon and uses the same configuration file. However, it runs in the foreground and exits after all input log data has been processed. Common input sources are files and databases. This tool is useful for log processing tasks such as:

- loading a group of files into a database,
- converting between different formats,
- testing patterns,
- doing offline event correlation, or
- checking HMAC message integrity.

While the details provided here apply to NXLog installations on Linux and other UNIX-style operating systems in particular, a few Windows-specific notes are included.

OPTIONS

-c *conffile*, --conf *conffile*

Specify an alternate configuration file *conffile*.

-h, --help

Print help.

-v, --verify

Verify configuration file syntax.

FILES

/opt/nxlog/bin/nxlog-processor

The main NXLog-processor executable

/opt/nxlog/bin/nxlog-stmnt-verifier

This tool can be used to check NXLog Language statements. All statements are read from standard input and

then validated. If a statement is invalid, the tool prints an error to standard error and exits non-zero.

/opt/nxlog/etc/nxlog.conf

The default configuration file

/opt/nxlog/spool/nxlog/configcache.dat

This is the position cache file where positions are saved. To disable position caching, as may be desirable when using nxlog-processor, set the **NoCache** directive to TRUE.

ENVIRONMENT

To access environment variables in the NXLog configuration, use the **envvar** directive.

SEE ALSO

[nxlog\(8\)](#)

NXLog website: <https://nxlog.co>

NXLog User Guide: <https://nxlog.co/documentation/nxlog-user-guide>

COPYRIGHT

Copyright © NXLog Ltd. 2018

The NXLog Community Edition is licensed under the NXLog Public License. The NXLog Enterprise Edition is not free and has a commercial license.

Chapter 84. Configuration

An NXLog configuration consists of global directives, module instances, and routes. The following sections list the core NXLog directives provided. Additional directives are provided at the module level.

A valid configuration must contain at least one input module instance and at least one output module instance. If no route is specified, a route will be automatically generated; this route will connect all input module instances and all output module instances in a single path.

84.1. General Directives

The following directives can be used throughout the configuration file. These directives are handled by the configuration parser, and substitutions occur before the configuration check.

define

Use this directive to configure a constant or macro to be used later. Refer to a **define** by surrounding the name with percent signs (%). Enclose a group of statements with curly braces ({}).

Example 332. Using the define Directive

This configuration shows three example defines: **BASEDIR** is a constant, **IMPORTANT** is a statement, and **WARN_DROP** is a group of statements.

nxlog.conf

```
1 define BASEDIR /var/log
2 define IMPORTANT if $raw_event =~ /important/ \
3                     $Message = 'IMPORTANT ' + $raw_event;
4 define WARN_DROP { log_warning("dropping message"); drop(); }
5
6 <Input messages>
7     Module  im_file
8     File    '%BASEDIR%/messages'
9 </Input>
10
11 <Input proftpd>
12     Module  im_file
13     File    '%BASEDIR%/proftpd.log'
14     <Exec>
15         %IMPORTANT%
16         if $raw_event =~ /dropme/ %WARN_DROP%
17     </Exec>
18 </Input>
```

envvar

This directive works like **define**, except that the value is retrieved from the environment.

Example 333. Using the envvar Directive

This example is like [the previous one](#), but **BASEDIR** is fetched from the environment instead.

nxlog.conf

```
1 envvar BASEDIR
2
3 <Input in>
4   Module im_file
5   File  '%BASEDIR%/messages'
6 </Input>
```

include

This directive allows a specified file to be included in the current configuration file. Wildcarded filenames are supported.

NOTE

The **SpoolDir** directive only takes effect after the configuration is parsed, so relative paths specified with the **include** directive must be relative to the working directory NXLog was started from.

Example 334. Using the include Directive

This example includes a file relative to the directory NXLog is started from:

```
include modules/module1.conf
```

This example includes all matching files and uses an absolute path:

```
include /etc/nxlog.d/*.conf
```

include_stdout

This directive accepts the name of an external command or script. Configuration content will be read from the command's standard output. Command arguments are not supported.

Example 335. Using the include_stdout Directive

This directive executes the custom script, which fetches the configuration.

nxlog.conf

```
1 include_stdout /opt/nxset/etc/fetch_conf.sh
```

84.2. Global Directives

CacheDir

This directive specifies a directory where the cache file (**configcache.dat**) should be written. This directive has a compiled-in value which is used by default.

CacheFlushInterval

This directive takes an integer or the string **always** as an argument. The cache file is flushed to disk at the interval, in seconds, specified here. When **always** is specified, the cache file is flushed immediately after a module sets a value. The default value of **0** indicates that the cache should only be flushed to the file on shutdown. During a server crash this value will be lost if the cache is not written to disk, so there is a compromise between performance and reliability.

CacheSync

When this directive is set to TRUE, the cache file is synced to disk after it is written. Otherwise the write operation will return even if the data is only in the system buffers and not yet on disk. Setting this to TRUE when [CacheFlushInterval](#) is set to [always](#) greatly reduces performance, though only this yields a reliable and crash-safe operation.

DateFormat

This directive can be used to change the default date format as it appears in the [LogFile](#), in the [\\$raw_event](#) generated by the modules, and when a [datetime](#) type value is converted to a string. The following values are accepted (corresponding to the formats accepted by the NXLog [strftime\(\)](#) function):

- `YYYY-MM-DD hh:mm:ss` (the default)
- `YYYY-MM-DDThh:mm:ssTZ`
- `YYYY-MM-DDThh:mm:ss.sTZ`
- `YYYY-MM-DD hh:mm:ssTZ`
- `YYYY-MM-DD hh:mm:ss.sTZ`
- `YYYY-MM-DDThh:mm:ssUTC`
- `YYYY-MM-DDThh:mm:ss.sUTC`
- `YYYY-MM-DD hh:mm:ssUTC`
- `YYYY-MM-DD hh:mm:ss.sUTC`
- A format string accepted by the `strftime()` function in the C library

FlowControl

This optional boolean directive specifies whether all input and processor modules should use flow control. This defaults to TRUE. See the description of the module level [FlowControl](#) directive for more information.

Group

Similar to [User](#), NXLog will set the group ID to run under. The group can be specified by name or numeric ID. This directive has no effect when running on the Windows platform or with [nxlog-processor\(8\)](#).

IgnoreErrors

If set to FALSE, NXLog will stop when it encounters a problem with the configuration file (such as an invalid module directive) or if there is any other problem which would prevent all modules functioning correctly. If set to TRUE, NXLog will start after logging the problem. The default value is TRUE.

LogFile

NXLog will write its internal log to this file. If this directive is not specified, self logging is disabled. Note that the [im_internal](#) module can also be used to direct internal log messages to files or different output destinations, but this does not support log level below [INFO](#). This [LogFile](#) directive is especially useful for debugging.

LogLevel

This directive has five possible values: [CRITICAL](#), [ERROR](#), [WARNING](#), [INFO](#), and [DEBUG](#). It will set both the logging level used for [LogFile](#) and the standard output if NXLog is started in the foreground. The default [LogLevel](#) is [INFO](#).

LogqueueDir

This directive specifies the directory where the files of the persistent queues are stored. This is only valid if [PersistLogqueue](#) is set to TRUE. If not specified, the default is the value of [CacheDir](#). This directive can also be used at the [module level](#).

LogqueueSize

When a log queue is full, the flow control in NXLog will pause the preceding module (an input module will pause after an associated output module's queue is full). The size of the queue is measured in event records, and can be set with this directive. The default is 100 records. This directive can also be used at the [module level](#).

ModuleDir

By default the NXLog binaries have a compiled-in value for the directory to search for loadable modules. This can be overridden with this directive. The module directory contains sub-directories for each module type (extension, input, output, and processor), and the module binaries are located in those.

NoCache

Some modules save data to a cache file which is persisted across a shutdown/restart. Modules such as [im_file](#) will save the file position in order to continue reading from the same position after a restart as before. This caching mechanism can be explicitly turned off with this directive. This is mostly useful with [nxlog-processor\(8\)](#) in offline mode. If this boolean directive is not specified, it defaults to FALSE (caching is enabled).

NoFreeOnExit

This directive is for debugging. When set to TRUE, NXLog will not free module resources on exit, allowing valgrind to show proper stack trace locations in module function calls. The default value is FALSE.

Panic

A panic condition is a critical state which usually indicates a bug. Assertions are used in NXLog code for checking conditions where the code will not work unless the asserted condition is satisfied, and for security. Failing assertions result in a panic and suggest a bug in the code. A typical case is checking for NULL pointers before pointer dereference. This directive can take three different values: **HARD**, **SOFT**, or **OFF**. **HARD** will cause an abort in case the assertion fails. This is how most C based programs work. **SOFT** will cause an exception to be thrown at the place of the panic/assertion. In case of NULL pointer checks this is identical to a NullPointerException in Java. It is possible that NXLog can recover from exceptions and can continue to process log messages, or at least the other modules can. In case of assertion failure the location and the condition is printed at **CRITICAL** log level in **HARD** mode and **ERROR** log level in **SOFT** mode. If **Panic** is set to **OFF**, the failing condition is printed in the logs but the execution will continue on the normal code path. Most of the time this will result in a segmentation fault or other undefined behavior, though in some cases turning off a buggy assertion or panic will solve the problems caused by it in **HARD/SOFT** mode. The default value for **Panic** is **SOFT**.

PersistLogqueue

When a module passes data to the next module along the route, it puts this into the next module's queue. This queue can be either a memory-based or disk-based (persistent) queue. When this directive is set to TRUE, all modules will use persistent (disk-based) incoming log queues. With the default value of FALSE, log queues are not persistent (are memory-based). This directive can also be used at the [module level](#).

PidFile

Under Unix operating systems, NXLog writes a PID file as other system daemons do. The default PID file can be overridden with this directive in case multiple daemon instances need to be running. This directive has no effect when running on the Windows platform or with [nxlog-processor\(8\)](#).

RootDir

NXLog will set its root directory to the value specified with this directive. If **SpoolDir** is also set, this will be relative to the value of **RootDir** (`chroot()` is called first). This directive has no effect when running on the Windows platform or with the [nxlog-processor\(8\)](#).

SpoolDir

NXLog will change its working directory to the value specified with this directive. This is useful with files created through relative filenames (for example, with [om_file](#)) and in case of core dumps. This directive has no effect with the [nxlog-processor\(8\)](#).

StringLimit

To protect against memory exhaustion (and possibly a denial-of-service) caused by over-sized strings, there is a limit on the length of each string (in bytes). The default value is **5242880** (strings will be truncated at 5 MiB).

SuppressRepeatingLogs

Under some circumstances it is possible for NXLog to generate an extreme amount of internal logs consisting of the same message due to an incorrect configuration or a software bug. In this case, the [LogFile](#) can quickly consume the available disk space. With this directive, NXLog will write at most 2 lines per second if the same message is generated successively, by logging "last message repeated n times" messages. If this boolean directive is not specified, it defaults to TRUE (suppression of repeating messages is enabled).

SyncLogqueue

When this directive is set to TRUE and [PersistLogqueue](#) is enabled, the disk-based queues will be immediately synced after each new entry is added to the queue. This greatly reduces performance but makes the queue more reliable and crash-safe. This directive can also be used at the [module level](#).

Threads

This directive specifies the number of worker threads to use. The number of the worker threads is calculated and set to an optimal value if this directive is not defined. Do not set this unless you know what you are doing.

User

NXLog will drop to the user specified with this directive. This is useful if NXLog needs privileged access to some system resources (such as kernel messages or to bind a port below 1024). On Linux systems NXLog will use capabilities to access these resources. In this case NXLog must be started as root. The user can be specified by name or numeric ID. This directive has no effect when running on the Windows platform or with [nxlog-processor\(8\)](#).

84.3. Common Module Directives

The following directives are common to all modules. The [Module](#) directive is mandatory.

Module

This mandatory directive specifies which binary should be loaded. The module binary has a **.so** extension on Unix and a **.dll** on Windows platforms and resides under the [ModuleDir](#) location. Each module binary name is prefixed with **im_**, **pm_**, **om_**, or **xm_** (for *input*, *processor*, *output*, and *extension*, respectively). It is possible for multiple instances to use the same loadable binary. In this case the binary is only loaded once but instantiated multiple times. Different module instances may have different configurations.

BufferSize

This directive specifies the size of the buffer used by the module, in bytes. The **BufferSize** directive can only be used with Input and Output modules, for all other types it is ignored. The default buffer size is 65000 bytes.

FlowControl

This optional boolean directive specifies whether the module should use flow control. This can only be used in Input and Processor modules. By default, **FlowControl** is TRUE (enabled). This module-level directive can be used to override the global [FlowControl](#) directive.

When flow control is in effect, a module (Input or Processor) which tries to forward log data to the next module in the route will be suspended if the next module cannot accept more data. For example, if a network module (such as [om_tcp](#)) cannot forward logs because of a network error, the preceding module in the route will be paused. When flow control is disabled, the module will drop the log record if the queue of the next module in the route is full.

Disabling flow control can be useful when multiple output modules are configured to store or forward log data. When flow control is enabled, the output modules will only process log data if all outputs are functional. Consider the case where log data is stored in a file using [om_file](#) and also forwarded over the network using [om_tcp](#). When flow control is enabled, a network disconnection will make the data flow stall and log data will not be written into the local file either. With flow control disabled, NXLog will write log data to the file and will drop messages that cannot be forwarded over the network.

WARNING

When using the [im_uds](#) module to collect local Syslog messages from the /dev/log Unix domain socket, **FlowControl** should be disabled. Otherwise, if the corresponding Output queue becomes full, the `syslog()` system call will block in any programs trying to write to the system log and an unresponsive system may result.

InputType

This directive specifies the name of the registered input reader function to be used for parsing raw events from input data. Names are treated case insensitively. This directive is only available for stream oriented input modules: [im_file](#), [im_exec](#), [im_ssl](#), [im_tcp](#), [im_udp](#), and [im_uds](#). These modules work by filling an input buffer with data read from the source. If the read operation was successful (there was data coming from the source), the module calls the specified callback function. If this is not explicitly specified, the module default will be used. Note that [im_udp](#) may only work properly if log messages do not span multiple packets and are within the UDP message size limit. Otherwise the loss of a packet may lead to parsing errors.

Modules may provide custom input reader functions. Once these are registered into the NXLog core, the modules listed above will be capable of using these. This makes it easier to implement custom protocols because these can be developed without concern for the transport layer.

The following input reader functions are provided by the NXLog core:

Binary

The input is parsed in the NXLog binary format, which preserves the parsed fields of the event records. The [LineBased](#) reader will automatically detect event records in the binary NXLog format, so it is only recommended to configure *InputType* to **Binary** if compatibility with other logging software is not required.

Dgram

Once the buffer is filled with data, it is considered to be one event record. This is the default for the [im_udp](#) input module, since UDP Syslog messages arrive in separate packets.

LineBased

The input is assumed to contain event records separated by newlines. It can handle both CRLF (Windows) and LF (Unix) line-breaks. Thus if an LF (`\n`) or CRLF (`\r\n`) is found, the function assumes that it has reached the end of the event record. If the input begins with the UTF-8 byte order mark (BOM) sequence (`0xEF, 0xBB, 0xBF`), that sequence is automatically skipped.

Example 336. TCP Input Assuming NXLog Format

This configuration explicitly specifies the **Binary** *InputType*.

nxlog.conf

```
1 <Input tcp>
2   Module      im_tcp
3   Port        2345
4   InputType   Binary
5 </Input>
```

LogqueueDir

This directive specifies the directory where the files of the persistent queue are stored. This is only valid if [PersistLogqueue](#) is set to TRUE. If not specified, the default is the value of [CacheDir](#). This directive can be

used at the [global level](#) to affect all modules.

LogqueueSize

When a log queue is full, the flow control in NXLog will pause the preceding module (an input module will pause after an associated output module's queue is full). The size of the queue is measured in event records, and can be set with this directive. The default is 100 records. This directive can be used at the [global level](#) to affect all modules.

OutputType

This directive specifies the name of the registered output writer function to be used for formatting raw events when storing or forwarding output. Names are treated case insensitively. This directive is only available for stream oriented output modules: [om_file](#), [om_exec](#), [om_ssl](#), [om_tcp](#), [om_udp](#), and [om_uds](#). These modules work by filling the output buffer with data to be written to the destination. The specified callback function is called before the write operation. If this is not explicitly specified, the module default will be used.

Modules may provide custom output formatter functions. Once these are registered into the NXLog core, the modules listed above will be capable of using these. This makes it easier to implement custom protocols because these can be developed without concern for the transport layer.

The following output writer functions are provided by the NXLog core:

Binary

The output is written in the NXLog binary format which preserves parsed fields of the event records.

Dgram

Once the buffer is filled with data, it is considered to be one event record. This is the default for the [om_udp](#) output module, since UDP Syslog messages are sent in separate packets.

LineBased

The output will contain event records separated by newlines. The record terminator is CRLF ([\r\n](#)) on Windows and LF ([\n](#)) on Unix.

LineBased_CRLF

The output will contain event records separated by Windows style newlines where the record terminator is CRLF ([\r\n](#)).

LineBased_LF

The output will contain event records separated by Unix style newlines where the record terminator is LF ([\n](#)).

Example 337. TCP Output Sending Messages in NXLog Format

This configuration explicitly specifies the [Binary](#) *OutputType*.

nxlog.conf

```
1 <Output tcp>
2   Module    om_tcp
3   Port      2345
4   Host      localhost
5   OutputType Binary
6 </Output>
```

PersistLogqueue

When a module passes data to the next module along the route, it puts this into the next module's queue. This queue can be either a memory-based or disk-based (persistent) queue. When this directive is set to TRUE, the module will use a persistent (disk-based) queue. With the default value of FALSE, the module's

incoming log queue will not be persistent (will be memory-based). This directive can be used at the [global level](#) to affect all modules. Note that input modules do not have a queue, so this directive is only relevant for processor and output modules.

SyncLogqueue

When this directive is set to TRUE and [PersistLogqueue](#) is enabled, the disk-based queue will be immediately synced after each new entry is added to the queue. This greatly reduces performance but makes the queue more reliable and crash-safe. This directive can be used at the [global level](#) to affect all modules.

84.3.1. Exec

The **Exec** directive/block contains [statements](#) in the [NXLog language](#) which are executed when a module receives a log message. This directive is available in all [input](#), [processor](#), and [output](#) modules. It is not available in most [extension](#) modules because these do not handle log messages directly (the [xm_multiline](#) and [xm_rewrite](#) modules do provide **Exec** directives).

Example 338. Simple Exec Statement

This statement assigns a value to the `$Hostname` field in the event record.

```
Exec $Hostname = 'myhost';
```

Each directive must be on one line unless it contains a trailing backslash (\) character.

Example 339. Exec Statement Spanning Multiple Lines

This **if** statement uses line continuation to span multiple lines.

```
Exec if $Message =~ /something interesting/ \
      log_info("found something interesting"); \
else \
      log_debug("found nothing interesting");
```

More than one **Exec** directive or block may be specified. They are executed in the order of appearance. Each **Exec** directive must contain a full statement. Therefore it is not possible to split the lines in the previous example into multiple **Exec** directives. It is only possible to split the **Exec** directive if it contains multiple statements.

Example 340. Equivalent Use of Statements in Exec

This example shows two equivalent uses of the **Exec** directive.

```
Exec log_info("first"); \
log_info("second");
```

This produces identical behavior:

```
Exec log_info("first");
Exec log_info("second");
```

The **Exec** directive can also be used as a block. To use multiple statements spanning more than one line, it is recommended to use the [**<Exec>**](#) block instead. When using a block, it is not necessary to use the backslash (\) character for line continuation.

Example 341. Using the Exec Block

This example shows two equivalent uses of **Exec**, first as a *directive*, then as a *block*.

```
Exec    log_info("first"); \
        log_info("second");
```

The following **Exec block** is equivalent. Notice the backslash (\) is omitted.

```
1 <Exec>
2     log_info("first");
3     log_info("second");
4 </Exec>
```

84.3.2. Schedule

The Schedule block can be used to execute periodic jobs, such as log rotation or any other task. Scheduled jobs have the same priority as the module. The Schedule block has the following directives:

Every

In addition to the crontab format it is possible to schedule execution at periodic intervals. With the crontab format it is not possible to run a job every five days for example, but this directive enables it in a simple way. It takes an integer value with an optional unit. The unit can be one of the following: **sec**, **min**, **hour**, **day**, or **week**. If the unit is not specified, the value is assumed to be in seconds.

Exec

The mandatory **Exec** directive takes one or more NXLog [statements](#). This is the code which is actually being scheduled. Multiple **Exec** directives can be specified within one **Schedule** block. See the module-level **Exec** directive, this behaves the same. Note that it is not possible to use **fields** in statements here because execution is not triggered by log messages.

First

This directive sets the first execution time. If the value is in the past, the next execution time is calculated as if NXLog has been running since and jobs will not be run to make up for missed events in the past. The directive takes a **datetime** literal value.

RunCount

This optional directive can be used to specify a maximum number of times that the corresponding **Exec** statement(s) should be executed. For example, with **RunCount 1** the statement(s) will only be executed once.

When

This directive takes a value similar to a crontab entry: five space-separated definitions for minute, hour, day, month, and weekday. See the crontab(5) manual for the field definitions. It supports lists as comma separated values and/or ranges. Step values are also supported with the slash. Month and week days are not supported, these must be defined with numeric values. The following extensions are also supported:

@yearly	Run once a year, "0 0 1 1 *".
@annually	(same as @yearly)
@monthly	Run once a month, "0 0 1 * *".
@weekly	Run once a week, "0 0 * * 0".
@daily	Run once a day, "0 0 * * *".
@midnight	(same as @daily)
@hourly	Run once an hour, "0 * * * *".

Example 342. Scheduled Exec Statements

This example shows two scheduled **Exec** statements in a **im_tcp** module instance. The first is executed every second, while the second uses a crontab(5) style value.

nxlog.conf

```
1 <Input in>
2   Module im_tcp
3   Port   2345
4
5   <Schedule>
6     Every 1 sec
7     First  2010-12-17 00:19:06
8     Exec   log_info("scheduled execution at " + now());
9   </Schedule>
10
11  <Schedule>
12    When   1 */2 2-4 * *
13    Exec   log_info("scheduled execution at " + now());
14  </Schedule>
15 </Input>
```

84.4. Route Directives

The following directives can be used in Route blocks. The **Path** directive is mandatory.

Path

The data flow is defined by the **Path** directive. First the instance names of Input modules are specified. If more than one Input reads log messages which feed data into the route, then these must be separated by commas. The list of Input modules is followed by an arrow (**=>**). Either processor modules or output modules follow. Processor modules must be separated by arrows, not commas, because they operate in series, unlike Input and Output modules which work in parallel. Output modules are separated by commas. The **Path** must specify at least an Input and an Output. The syntax is illustrated by the following:

```
Path INPUT1[ , INPUT2... ] => [PROCESSOR1 [=] PROCESSOR2...] =>] OUTPUT1[ , OUTPUT2... ]
```

Example 343. Specifying Routes

The following configuration shows modules being used in two routes.

nxlog.conf

```
1 <Input in1>
2   Module im_null
3 </Input>
4
5 <Input in2>
6   Module im_null
7 </Input>
8
9 <Processor p1>
10  Module pm_null
11 </Processor>
12
13 <Processor p2>
14  Module pm_null
15 </Processor>
16
17 <Output out1>
18  Module om_null
19 </Output>
20
21 <Output out2>
22  Module om_null
23 </Output>
24
25 <Route 1>
26   # Basic route
27   Path in1 => out1
28 </Route>
29
30 <Route 2>
31   # Complex route with multiple input/output/processor modules
32   Path in1, in2 => p1 => p2 => out1, out2
33 </Route>
```

Priority

This directive takes an integer value in the range of 1-100 as a parameter, and the default is **10**. Log messages in routes with a lower **Priority** value will be processed before others. Internally, this value is assigned to each module part of the route. The events of the modules are processed in priority order by the NXLog engine. Modules of a route with a lower **Priority** value (higher priority) will process log messages first.

Example 344. Prioritized Processing

This configuration prioritizes the UDP route over the TCP route in order to minimize loss of UDP Syslog messages when the system is busy.

nxlog.conf

```
1 <Input tcpin>
2   Module      im_tcp
3   Host        localhost
4   Port        514
5 </Input>
6
7 <Input udpin>
8   Module      im_udp
9   Host        localhost
10  Port       514
11 </Input>
12
13 <Output tcpfile>
14   Module      om_file
15   File        "/var/log/tcp.log"
16 </Output>
17
18 <Output udpfile>
19   Module      om_file
20   File        "/var/log/udp.log"
21 </Output>
22
23 <Route udp>
24   Priority    1
25   Path        udpin => udpfile
26 </Route>
27
28 <Route tcp>
29   Priority    2
30   Path        tcpin => tcpfile
31 </Route>
```

Chapter 85. Language

85.1. Types

The following types are provided by the NXLog language.

Unknown

This is a special type for values where the type cannot be determined at compile time and for uninitialized values. The [undef literal](#) and [fields](#) without a value also have an unknown type. The unknown type can also be thought of as "any" in case of function and procedure API declarations.

Boolean

A boolean value is TRUE, FALSE or undefined. Note that an undefined value is not the same as a FALSE value.

Integer

An integer can hold a signed 64 bit value in addition to the undefined value. Floating point values are not supported.

String

A string is an array of characters in any character set. The [binary](#) type should be used for values where the NUL byte can also occur. An undefined string is not the same as an empty string. Strings have a limited length to prevent resource exhaustion problems, this is a compile-time value currently set to 1M.

Datetime

A datetime holds a microsecond value of time elapsed since the Epoch. It is always stored in UTC/GMT.

IPv4 Address

An ip4addr type stores a dotted-quad IPv4 address in an internal format (integer).

IPv6 Address

An ip6addr type stores an IPv6 address in an internal format.

Regular expression

A regular expression type can only be used with the `=~` or `!~` operators.

Binary

This type can hold an array of bytes.

Variadic arguments

This is a special type only used in function and procedure API declarations to indicate variadic arguments.

85.2. Expressions

85.2.1. Literals

Undef

The undef literal has an [unknown](#) type. It can be also used in an [assignment](#) to unset the value of a [field](#).

Example 345. Un-Setting the Value of a Field

This statement unsets the `$ProcessID` field.

```
1 $ProcessID = undef;
```

Boolean

A boolean literal is either TRUE or FALSE. It is case-insensitive, so **True**, **False**, **true**, and **false** are also valid.

Integer

An integer starts with a minus (-) sign if it is negative. A "0X" or "0x" prepended modifier indicates a hexadecimal notation. The "K", "M" and "G" modifiers are also supported; these mean Kilo (1024), Mega (1024²), or Giga (1024³) respectively when appended.

Example 346. Setting an Integer Value

This statement uses a modifier to set the **\$Limit** field to 44040192 (42×1024²).

```
1 $Limit = 42M;
```

String

String literals are quoted characters using either single or double quotes. String literals specified with double quotes can contain the following escape sequences.

\\

The backslash (\) character.

\"

The double quote ("") character.

\n

Line feed (LF).

\r

Carriage return (CR).

\t

Horizontal tab.

\b

Audible bell.

\xXX

A single byte in the form of a two digit hexadecimal number. For example the line-feed character can also be expressed as **\x0A**.

NOTE String literals in single quotes do not process the escape sequences: "\n" is a single character (LF) while '\n' is two characters. The following comparison is FALSE for this reason: "\n" == '\n'.

NOTE Extra care should be taken with the backslash when using double quoted string literals to specify file paths on Windows. See [this note](#) for the file directive of im_file about the possible complications.

Example 347. Setting a String Value

This statement sets the **\$Message** field to the specified string.

```
1 $Message = "Test message";
```

Datetime

A datetime literal is an unquoted representation of a time value expressing local time in the format of **YYYY-MM-DD hh:mm:ss**.

Example 348. Setting a Datetime Value

This statement sets the **\$EventTime** field to the specified datetime value.

```
1 $EventTime = 2000-01-02 03:04:05;
```

IPv4 Address

An IPv4 literal value is expressed in dotted quad notation such as **192.168.1.1**.

IPv6 Address

An IPv6 literal value is expressed by 8 groups of 16-bit hexadecimal values separated by colons (:) such as **2001:0db8:85a3:0000:0000:8a2e:0370:7334**.

85.2.2. Regular Expressions

The **PCRE** engine is used to execute regular expressions in NXLog. For more information about the PCRE syntax, see the [pcre2syntax\(3\)](#) and [pcre2pattern\(3\)](#) man pages.

Regular expressions must be used with one of the **=~** and **!~** operators, and must be quoted with slashes (/) as in Perl. Captured sub-strings are accessible through numeric reference, and the full subject string is placed into **\$0**.

Example 349. A Regular Expression Match Operation

If the regular expression matches the **\$Message** field, the **log_info()** procedure is executed. The captured sub-string is used in the logged string (**\$1**).

```
1 if $Message =~ /^Test (\$+)/ log_info("captured: " + $1);
```

It is also possible to use named capturing such that the resulting field name is defined in the regular expression.

Example 350. Regular Expression Match Using Named Capturing

This statement causes the same behavior as the previous example, but it uses named capturing instead.

```
1 if $Message =~ /Test: (?<test>\$+)/ log_info("captured: " + $test);
```

Substitution is supported with the **s///** operator. Variables and captured sub-string references cannot be used inside the regular expression or the substitution operator (they will be treated literally).

85.2.2.1. Regular Expression Modifiers

The following regular expression modifiers are supported:

g

The **/g** modifier can be used for global replacement.

Example 351. Replace Whitespace Occurrences

If any whitespace is found in the **\$SourceName** field, it is replaced with underscores (_) and a log message is generated.

```
1 if $SourceName =~ s/\s/_/g log_info("removed all whitespace in SourceName");
```

s

The dot (.) normally matches any character except newline. The /s modifier causes the dot to match all characters including line terminator characters (LF and CRLF).

Example 352. Dot Matches All Characters

The regular expression in this statement will match a message that begins and ends with the given keywords, even if the message spans multiple lines.

```
1 if $Message =~ /^Backtrace.*END$/s drop();
```

m

The /m modifier can be used to treat the string as multiple lines (^ and \$ match newlines within data).

i

The /i modifier does case insensitive matching.

85.2.3. Fields

Fields are referenced in the NXLog language by prepending a dollar sign (\$) to the field name. A field name can contain the characters [a-zA-Z0-9_] but must begin with a letter or underscore (_), as indicated by the following regular expression:

```
[[[:alpha:]_][[:alnum:]\_]*
```

Fields containing special characters such as the space or minus (-) can be specified using curly braces such as \${file-size} or \${file size}.

A field which does not exist has an **unknown** type.

85.2.4. Operations

85.2.4.1. Unary Operations

The following unary operations are available. It is possible to use brackets around the operand to make it look like a function call as in the "defined" example below.

not

The **not** operator expects a boolean value. It will evaluate to undef if the value is undefined. If it receives an unknown value which evaluates to a non-boolean, it will result in a run-time execution error.

Example 353. Using the "not" Operand

If the **\$Success** field has a value of false, an error is logged.

```
1 if not $Success log_error("Job failed");
```

defined

The defined operator will evaluate to TRUE if the operand is defined, otherwise FALSE.

Example 354. Using the Unary "defined" Operation

This statement is a no-op, it does nothing.

```
1 if defined undef log_info("never printed");
```

If the **\$EventTime** field has not been set (due perhaps to failed parsing), it will be set to the current time.

```
1 if not defined($EventTime) $EventTime = now();
```

85.2.4.2. Binary Operations

The following binary operations are available.

The operations are described with the following syntax:

LEFT_OPERAND_TYPE OPERATION RIGHT_OPERAND_TYPE = EVALUATED_VALUE_TYPE

=~

This is the regular expression match operation as in Perl. This operation takes a string and a regular expression operand and evaluates to a boolean value which will be TRUE if the regular expression matches the subject string. Captured sub-strings are accessible through numeric reference (such as **\$1**) and the full subject string is placed into **\$0**. Regular expression based string substitution is supported with the **s///** operator. For more details, see [Regular Expressions](#).

- **string =~ regexp = boolean**
- **regexp =~ string = boolean**

Example 355. Regular Expression Based String Matching

A log message will be generated if the **\$Message** field matches the regular expression.

```
1 if $Message =~ /^Test message/ log_info("matched");
```

!~

This is the opposite of **=~**: the expression will evaluate to TRUE if the regular expression does not match on the subject string. It can be also written as **not LEFT_OPERAND =~ RIGHT_OPERAND**. The **s///** substitution operator is supported.

- **string !~ regexp = boolean**
- **regexp !~ string = boolean**

Example 356. Regular Expression Based Negative String Matching

A log message will be generated if the **\$Message** field does not match the regular expression.

```
1 if $Message !~ /^Test message/ log_info("didn't match");
```

==

This operator compares two values for equality. Comparing a defined value with an undefined results in

`undef.`

- `undef == undef = TRUE`
- `string == string = boolean`
- `integer == integer = boolean`
- `boolean == boolean = boolean`
- `datetime == datetime = boolean`
- `ip4addr == ip4addr = boolean`
- `ip4addr == string = boolean`
- `string == ip4addr = boolean`

Example 357. Equality

A log message will be generated if `$SeverityValue` is 1.

```
1 if $SeverityValue == 1 log_info("severity is one");
```

`!=`

This operator compares two values for inequality. Comparing a defined value with an undefined results in `undef`.

- `undef != undef = FALSE`
- `string != string = boolean`
- `integer != integer = boolean`
- `boolean != boolean = boolean`
- `datetime != datetime = boolean`
- `ip4addr != ip4addr = boolean`
- `ip4addr != string = boolean`
- `string != ip4addr = boolean`

Example 358. Inequality

A log message will be generated if `$SeverityValue` is not 1.

```
1 if $SeverityValue != 1 log_info("severity is not one");
```

`<`

This operation will evaluate to TRUE if the left operand is less than the right operand, and FALSE otherwise. Comparing a defined value with an undefined results in `undef`.

- `integer < integer = boolean`
- `datetime < datetime = boolean`

Example 359. Less

A log message will be generated if **\$SeverityValue** is less than 1.

```
1 if $SeverityValue < 1 log_info("severity is less than one");
```

<=

This operation will evaluate to TRUE if the left operand is less than or equal to the right operand, and FALSE otherwise. Comparing a defined value with an undefined results in **undef**.

- **integer <= integer = boolean**
- **datetime <= datetime = boolean**

Example 360. Less or Equal

A log message will be generated if **\$SeverityValue** is less than or equal to 1.

```
1 if $SeverityValue < 1 log_info("severity is less than or equal to one");
```

>

This operation will evaluate to TRUE if the left operand is greater than the right operand, and FALSE otherwise. Comparing a defined value with an undefined results in **undef**.

- **integer > integer = boolean**
- **datetime > datetime = boolean**

Example 361. Greater

A log message will be generated if **\$SeverityValue** is greater than 1.

```
1 if $SeverityValue > 1 log_info("severity is greater than one");
```

>=

This operation will evaluate to TRUE if the left operand is greater than or equal to the right operand, and FALSE otherwise. Comparing a defined value with an undefined results in **undef**.

- **integer >= integer = boolean**
- **datetime >= datetime = boolean**

Example 362. Greater or Equal

A log message will be generated if **\$SeverityValue** is greater than or equal to 1.

```
1 if $SeverityValue >= 1 log_info("severity is greater than or equal to one");
```

and

This operation evaluates to TRUE if and only if both operands are TRUE. The operation will evaluate to **undef** if either operand is undefined.

boolean and boolean = boolean

Example 363. And Operation

A log message will be generated only if both **\$SeverityValue** equals 1 and **\$FacilityValue** equals 2.

```
1 if $SeverityValue == 1 and $FacilityValue == 2 log_info("1 and 2");
```

or

This operation evaluates to TRUE if either operand is TRUE. The operation will evaluate to **undef** if both operands are undefined.

boolean or boolean = boolean

Example 364. Or Operation

A log message will be generated if **\$SeverityValue** is equal to either 1 or 2.

```
1 if $SeverityValue == 1 or $SeverityValue == 2 log_info("1 or 2");
```

+

This operation will result in an integer if both operands are integers. If either operand is a string, the result will be a string where non-string typed values are converted to strings. In this case it acts as a concatenation operator, like the dot (.) operator in Perl. Adding an undefined value to a non-string will result in undef.

- **integer + integer = integer**
- **string + undef = string**
- **undef + string = string**
- **undef + undef = undef**
- **string + string = string** (Concatenate two strings.)
- **datetime + integer = datetime** (Add the number of seconds in the right value to the datetime stored in the left value.)
- **integer + datetime = datetime** (Add the number of seconds in the left value to the datetime stored in the right value.)

Example 365. Concatenation

This statement will always cause a log message to be generated.

```
1 if 1 + "a" == "1a" log_info("this will be printed");
```

Subtraction. The result will be undef if either operand is undefined.

- **integer - integer = integer** (Subtract two integers.)
- **datetime - datetime = integer** (Subtract two datetime types. The result is the difference between two expressed in microseconds.)
- **datetime - integer = datetime** (Subtract the number of seconds from the datetime stored in the left value.)

Example 366. Subtraction

This statement will always cause a log message to be generated.

```
1 if 4 - 1 == 3 log_info("four minus one is three");
```

*

Multiply an integer with another. The result will be `undef` if either operand is `undefined`.

integer * integer = integer*Example 367. Multiplication*

This statement will always cause a log message to be generated.

```
1 if 4 * 2 == 8 log_info("four times two is eight");
```

/

Divide an integer with another. The result will be `undef` if either operand is `undefined`. Since the result is an integer, a fractional part is lost.

integer / integer = integer*Example 368. Division*

This statement will always cause a log message to be generated.

```
1 if 9 / 4 == 2 log_info("9 divided by 4 is 2");
```

%

The modulo operation divides an integer with another and returns the remainder. The result will be `undef` if either operand is `undefined`.

integer % integer = integer*Example 369. Modulo*

This statement will always cause a log message to be generated.

```
1 if 3 % 2 == 1 log_info("three mod two is one");
```

IN

This operation will evaluate to `TRUE` if the left operand is equal to any of the expressions in the list on the right, and `FALSE` otherwise. Comparing a `undefined` value results in `undef`.

unknown IN unknown, unknown ... = boolean*Example 370. IN*

A log message will be generated if `$EventID` is equal to any one of the values in the list.

```
1 if $EventID IN (1000, 1001, 1004, 4001) log_info("EventID found");
```

NOT IN

This operation is equivalent to `NOT expr IN expr_list`.

```
unknown NOT IN unknown, unknown ... = boolean
```

Example 371. NOT IN

A log message will be generated if `$EventID` is not equal to any of the values in the list.

```
1 if $EventID NOT IN (1000, 1001, 1004, 4001) log_info("EventID not in list");
```

85.2.4.3. Ternary Operation

The ternary operator `expr1 ? expr2 : expr3` evaluates to `expr2` if `expr1` is TRUE, otherwise to `expr3`. The parentheses as shown here are optional.

Example 372. Using the Ternary Operator

The `$Important` field is set to TRUE if `$SeverityValue` is greater than 2, or FALSE otherwise.

```
1 $Important = ( $SeverityValue > 2 ? TRUE : FALSE );
```

85.2.5. Functions

See [Functions](#) for a list of functions provided by the NXLog core. Additional functions are available through modules.

Example 373. A Function Call

This statement uses the `now()` function to set the field to the current time.

```
1 $EventTime = now();
```

It is also possible to call a function of a specific module instance.

Example 374. Calling a Function of a Specific Module Instance

This statement calls the `file_name()` and `file_size()` functions of a defined `om_file` instance named `out` in order to log the name and size of its currently open output file.

```
1 log_info('Size of output file ' + out->file_name() + ' is ' + out->file_size());
```

85.3. Statements

The following elements can be used in statements. There is no loop operation (`for` or `while`) in the NXLog language.

85.3.1. Assignment

The assignment operation is declared with an equal sign (`=`). It loads the value from the expression evaluated on the right into a `field` on the left.

Example 375. Field Assignment

This statement sets the **\$EventReceivedTime** field to the value returned by the **now()** function.

```
1 $EventReceivedTime = now();
```

85.3.2. Block

A block consists of one or more statements within curly braces (**{}**). This is typically used with [conditional statements](#) as in the example below.

Example 376. Conditional Statement Block

If the expression matches, both log messages will be generated.

```
1 if now() > 2000-01-01 00:00:00
2 {
3     log_info("we are in the");
4     log_info("21st century");
5 }
```

85.3.3. Procedures

See [Procedures](#) for a list of procedures provided by the NXLog core. Additional procedures are available through modules.

Example 377. A Procedure Call

The **log_info()** procedure generates an internal log message.

```
1 log_info("No log source activity detected.");
```

It is also possible to call a procedure of a specific module instance.

Example 378. Calling a Procedure of a Specific Module Instance

This statement calls the **parse_csv()** procedure of a defined *xm_csv* module instance named **csv_parser**.

```
1 csv_parser->parse_csv();
```

85.3.4. If-Else

A conditional statement starts with the **if** keyword followed by a boolean expression and a statement. The **else** keyword, followed by another statement, is optional. Brackets around the expression are also optional.

Example 379. Conditional Statements

A log message will be generated if the expression matches.

```
1 if now() > 2000-01-01 00:00:00 log_info("we are in the 21st century");
```

This statement is the same as the previous, but uses brackets.

```
1 if ( now() > 2000-01-01 00:00:00 ) log_info("we are in the 21st century");
```

This is a conditional statement block.

```
1 if now() > 2000-01-01 00:00:00
2 {
3     log_info("we are in the 21st century");
4 }
```

This conditional statement block includes an else branch.

```
1 if now() > 2000-01-01 00:00:00
2 {
3     log_info("we are in the 21st century");
4 }
5 else log_info("we are not yet in the 21st century");
```

Like Perl, the NXLog language does not have a *switch* statement. Instead, this can be accomplished by using conditional *if-else* statements.

Example 380. Emulating "switch" With "if-else"

The generated log message varies based on the value of the **\$value** field.

```
1 if ( $value == 1 )
2     log_info("1");
3 else if ( $value == 2 )
4     log_info("2");
5 else if ( $value == 3 )
6     log_info("3");
7 else
8     log_info("default");
```

NOTE The Perl *elsif* and *unless* keywords are not supported.

85.4. Variables

A module variable can only be accessed from the same module instance where it was created. A variable is referenced by a string value and can store a value of any type.

See the [create_var\(\)](#), [delete_var\(\)](#), [set_var\(\)](#), and [get_var\(\)](#) procedures.

85.5. Statistical Counters

The following types are available for statistical counters:

COUNT

Added values are aggregated, and the value of the counter is increased if only positive integers are added

until the counter is destroyed or indefinitely if the counter has no expiry.

COUNTMIN

This calculates the minimum value of the counter.

COUNTMAX

This calculates the maximum value of the counter.

AVG

This algorithm calculates the average over the specified interval.

AVGMIN

This algorithm calculates the average over the specified interval, and the value of the counter is always the lowest which was ever calculated during the lifetime of the counter.

AVGMAX

Like AVGMIN, but this returns the highest value calculated during the lifetime of the counter.

RATE

This calculates the value over the specified interval. It can be used to calculate events per second (EPS) values.

RATEMIN

This calculates the value over the specified interval, and returns the lowest rate calculated during the lifetime of the counter.

RATEMAX

Like RATEMIN, but this returns the highest rate calculated during the lifetime of the counter.

GRAD

This calculates the change of the rate of the counter over the specified interval, which is the gradient.

GRADMIN

This calculates the gradient and returns the lowest gradient calculated during the lifetime of the counter.

GRADMAX

Like GRADMIN, but this returns the highest gradient calculated during the lifetime of the counter.

85.6. Functions

The following functions are exported by core.

`datetime datetime(integer arg)`

Convert the integer argument, expressing the number of microseconds since epoch, to datetime.

`integer day(datetime datetime)`

Return the day part of the time value.

`integer dayofweek(datetime datetime)`

Return the number of days since Sunday in the range of 0-6.

`integer dayofyear(datetime datetime)`

Return the day number of the year in the range of 1-366.

`boolean dropped()`

Return TRUE if the currently processed event has already been dropped.

`string escape_html(string html)`

Return the HTML escaped *html* string.

`string escape_json(string jsonstr)`

Escape and return *jsonstr* according to the JSON specification.

`string escape_url(string url)`

Return the URL encoded string for *url*.

`string escape_xml(string xmlstr)`

Return the XML escaped *xmlstr* string.

`datetime fix_year(datetime datetime)`

Set the year value to the current year in a *datetime* which was parsed with a missing year, such as BSD Syslog or Cisco timestamps.

`integer get_rand()`

Return a random integer value.

`integer get_rand(integer max)`

Return a random integer value between 0 and *max*.

`integer get_sequence(string name)`

Return a number for the specified *sequence* that is incremented after each call to this function.

`integer get_stat(string statname)`

Return the value of the statistical counter or undef if it does not exist.

`integer get_stat(string statname, datetime time)`

Return the value of the statistical counter or undef if it does not exist. The *time* argument specifies the current time.

`string get_uuid()`

Return a UUID string.

`unknown get_var(string varname)`

Return the value of the variable or undef if it does not exist.

`ip4addr host_ip()`

Return the first non-loopback IP address the hostname resolves to.

`ip4addr host_ip(integer nth)`

Return the *nth* non-loopback IP address the hostname resolves to. The *nth* argument starts from 1.

`string hostname()`

Return the hostname (short form).

`string hostname_fqdn()`

Return the FQDN hostname. This function will return the short form if the FQDN hostname cannot be determined.

`integer hour(datetime datetime)`

Return the hour part of the time value.

integer integer(*unknown arg*)

Parse and convert the string argument to an integer. For datetime type it returns the number of microseconds since epoch.

ip4addr ip4addr(*integer arg*)

Convert the integer argument to an ip4addr type.

ip4addr ip4addr(*integer arg, boolean ntoa*)

Convert the integer argument to an ip4addr type. If *ntoa* is set to true, the integer is assumed to be in network byte order. Instead of **1.2.3.4** the result will be **4.3.2.1**.

string lc(*string arg*)

Convert the string to lower case.

integer microsecond(*datetime datetime*)

Return the microsecond part of the time value.

integer minute(*datetime datetime*)

Return the minute part of the time value.

integer month(*datetime datetime*)

Return the month part of the *datetime* value.

datetime now()

Return the current time.

string nxlog_version()

Return the NXLog version string.

datetime parsedate(*string arg*)

Parse a string containing a timestamp. Dates without timezone information are treated as local time. The current year is used for formats that do not include the year. An **undefined** datetime type is returned if the argument cannot be parsed, so that the user can fix the error (for example, **\$EventTime = parsedate(\$somestring); if not defined(\$EventTime) \$EventTime = now();**). Supported timestamp formats are listed below.

RFC 3164 (legacy Syslog) and variations

```
Nov 6 08:49:37
Nov 6 08:49:37
Nov 06 08:49:37
Nov 3 14:50:30.403
Nov 3 14:50:30.403
Nov 03 14:50:30.403
Nov 3 2005 14:50:30
Nov 3 2005 14:50:30
Nov 03 2005 14:50:30
Nov 3 2005 14:50:30.403
Nov 3 2005 14:50:30.403
Nov 03 2005 14:50:30.403
Nov 3 14:50:30 2005
Nov 3 14:50:30 2005
Nov 03 14:50:30 2005
```

RFC 1123

RFC 1123 compliant dates are also supported, including a couple others which are similar such as those

defined in RFC 822, RFC 850, and RFC 1036.

```
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994 ; ANSI C's asctime() format
Sun, 6 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sun, 06 Nov 94 08:49:37 GMT ; RFC 822
Sun, 6 Nov 94 08:49:37 GMT ; RFC 822
Sun, 06 Nov 94 08:49 GMT ; Unknown
Sun, 6 Nov 94 08:49 ; Unknown
Sun, 06 Nov 94 8:49:37 GMT ; Unknown [Elm 70.85]
Sun, 6 Nov 94 8:49:37 GMT ; Unknown [Elm 70.85]
Mon, 7 Jan 2002 07:21:22 GMT ; Unknown [Postfix]
Sun, 06-Nov-1994 08:49:37 GMT ; RFC 850 with four digit years
```

The above formats are also recognized when the leading day of week and/or the timezone are omitted.

Apache/NCSA date

This format can be found in Apache access logs and other sources.

```
24/Aug/2009:16:08:57 +0200
```

ISO 8601 and RFC 3339

NXLog can parse the ISO format with or without sub-second resolution, and with or without timezone information. It accepts either a comma (,) or a dot (.) in case there is sub-second resolution.

```
1977-09-06 01:02:03
1977-09-06 01:02:03.004
1977-09-06T01:02:03.004Z
1977-09-06T01:02:03.004+02:00
2011-5-29 0:3:21
2011-5-29 0:3:21+02:00
2011-5-29 0:3:21.004
2011-5-29 0:3:21.004+02:00
```

Windows timestamps

```
20100426151354.537875
20100426151354.537875-000
20100426151354.537875000
3/13/2017 8:42:07 AM ; Microsoft DNS Server
```

Integer timestamp

This format is **XXXXXXXXXX.USC**. The value is expressed as an integer showing the number of seconds elapsed since the epoch UTC. The fractional microsecond part is optional.

```
1258531221.650359
1258531221
```

BIND9 timestamps

```
23-Mar-2017 06:38:30.143
23-Mar-2017 06:38:30
2017-Mar-23 06:38:30.143
2017-Mar-23 06:38:30
```

string replace(string subject, string src, string dst)

Replace all occurrences of *src* with *dst* in the *subject* string.

string replace(string subject, string src, string dst, integer count)

Replace *count* number occurrences of *src* with *dst* in the *subject* string.

integer second(datetime datetime)

Return the second part of the time value.

integer size(string str)

Return the size of the string *str* in bytes.

string strftime(datetime datetime, string fmt)

Convert a datetime value to a string with the given format. The format must be one of:

- YYYY-MM-DD hh:mm:ss,
- YYYY-MM-DDThh:mm:ssTZ,
- YYYY-MM-DDThh:mm:ss.sTZ,
- YYYY-MM-DD hh:mm:ssTZ,
- YYYY-MM-DD hh:mm:ss.sTZ,
- YYYY-MM-DDThh:mm:ssUTC,
- YYYY-MM-DDThh:mm:ss.sUTC,
- YYYY-MM-DD hh:mm:ssUTC,
- YYYY-MM-DD hh:mm:ss.sUTC, or
- a format string accepted by the C strftime() function (see the strftime(3) manual or the Windows **strftime** reference for the format specification).

string string(unknown arg)

Convert the argument to a string.

datetime strptime(string input, string fmt)

Convert the string to a datetime with the given format. See the manual of strptime(3) for the format specification.

string substr(string src, integer from)

Return the string starting at the byte offset specified in *from*.

string substr(string src, integer from, integer to)

Return a sub-string specified with the starting and ending positions as byte offsets from the beginning of the string.

string type(unknown arg)

Return the type of the variable, which can be **boolean**, **integer**, **string**, **datetime**, **ip4addr**, **ip6addr**, **regexp**, or **binary**. For values with the unknown type, it returns **undef**.

string uc(string arg)

Convert the string to upper case.

string unescape_html(string html)

Return the HTML unescaped *html* string.

```
string unescape_json(string jsonstr)
```

Unescape and return *jsonstr* according to the JSON specification.

```
string unescape_url(string url)
```

Return the URL decoded string for *url*.

```
string unescape_xml(string xmlstr)
```

Return the XML unescaped *xmlstr* string.

```
integer year(datetime datetime)
```

Return the year part of the *datetime* value.

85.7. Procedures

The following procedures are exported by core.

```
add_stat(string statname, integer value);
```

Add *value* to the statistical counter using the current time.

```
add_stat(string statname, integer value, datetime time);
```

Add *value* to the statistical counter using the time specified in the argument named *time*.

```
add_to_route(string routename);
```

Copy the currently processed event data to the route specified. This procedure makes a copy of the data. The original will be processed normally. Note that flow control is explicitly disabled when moving data with *add_to_route()* and the data will not be added if the queue of the target module(s) is full.

```
create_stat(string statname, string type);
```

Create a module statistical counter with the specified name using the current time. The statistical counter will be created with an infinite lifetime. The *type* argument must be one of the following to select the required algorithm for calculating the value of the statistical counter: COUNT, COUNTMIN, COUNTMAX, AVG, AVGMIN, AVGMAX, RATE, RATEMIN, RATEMAX, GRAD, GRADMIN, or GRADMAX (see [Statistical Counters](#)).

This procedure with two parameters can only be used with COUNT, otherwise the interval parameter must be specified (see below). This procedure will do nothing if a counter with the specified name already exists.

```
create_stat(string statname, string type, integer interval);
```

Create a module statistical counter with the specified name to be calculated over *interval* seconds and using the current time. The statistical counter will be created with an infinite lifetime.

```
create_stat(string statname, string type, integer interval, datetime time);
```

Create a module statistical counter with the specified name to be calculated over *interval* seconds and the time value specified in the *time* argument. The statistical counter will be created with an infinite lifetime.

```
create_stat(string statname, string type, integer interval, datetime time, integer lifetime);
```

Create a module statistical counter with the specified name to be calculated over *interval* seconds and the time value specified in the *time* argument. The statistical counter will expire after *lifetime* seconds.

```
create_stat(string statname, string type, integer interval, datetime time, datetime expiry);
```

Create a module statistical counter with the specified name to be calculated over *interval* seconds and the time value specified in the *time* argument. The statistical counter will expire at *expiry*.

```
create_var(string varname);
```

Create a module variable with the specified name. The variable will be created with an infinite lifetime.

```
create_var(string varname, integer lifetime);
```

Create a module variable with the specified name and the *lifetime* given in seconds. When the lifetime expires, the variable will be deleted automatically and `get_var(name)` will return `undef`.

```
create_var(string varname, datetime expiry);
```

Create a module variable with the specified name. The *expiry* specifies when the variable should be deleted automatically.

```
debug(unknown arg, varargs args);
```

Print the argument(s) at DEBUG log level. Same as `log_debug()`.

```
delete(unknown arg);
```

Delete the field from the event. For example, `delete($field)`. Note that `$field = undef` is not the same, though after both operations the field will be undefined.

```
delete(string arg);
```

Delete the field from the event. For example, `delete("field")`.

```
delete_all();
```

Delete all of the fields from the event except `raw_event` field.

```
delete_stat(string statname);
```

Delete a module statistical counter with the specified name.

This procedure will do nothing if a counter with the specified name does not exist (eg. already deleted).

```
delete_var(string varname);
```

Delete the module variable with the specified name if it exists.

```
drop();
```

Drop the event record that is currently being processed. Any further action on the event record will result in a "missing logdata" error.

```
duplicate_guard();
```

Guard against event duplication.

```
log_debug(unknown arg, varargs args);
```

Print the argument(s) at DEBUG log level. Same as `debug()`.

```
log_error(unknown arg, varargs args);
```

Print the argument(s) at ERROR log level.

```
log_info(unknown arg, varargs args);
```

Print the argument(s) at INFO log level.

```
log_warning(unknown arg, varargs args);
```

Print the argument(s) at WARNING log level.

```
module_restart();
```

Issue `module_stop` and then a `module_start` events for the calling module.

```
module_start();
```

Issue a `module_start` event for the calling module.

```
module_stop();
```

Issue a `module_stop` event for the calling module.

```
rename_field(unknown old, unknown new);
```

Rename a field. For example, `rename_field($old, $new)`.

```
rename_field(string old, string new);
```

Rename a field. For example, `rename_field("old", "new")`.

```
reroute(string routename);
```

Move the currently processed event data to the route specified. The event data will enter the route as if it was received by an input module there. Note that flow control is explicitly disabled when moving data with `reroute()` and the data will be dropped if the queue of the target module(s) is full.

```
set_var(string varname, unknown value);
```

Set the value of a module variable. If the variable does not exist, it will be created with an infinite lifetime.

```
sleep(integer interval);
```

Sleep the specified number of microseconds. This procedure is provided for testing purposes primarily. It can be used as a poor man's rate limiting tool, though this use is not recommended.

Chapter 86. Extension Modules

Extension modules do not process log messages directly, and for this reason their instances cannot be part of a route. These modules enhance the features of NXLog in various ways, such as exporting new functions and procedures or registering additional I/O reader and writer functions (to be used with modules supporting the [InputType](#) and [OutputType](#) directives). There are many ways to hook an extension module into the NXLog engine, as the following modules illustrate.

86.1. Remote Management (`xm_admin`)

This module provides secure remote administration capabilities for the NXLog engine using either SOAP over HTTP/HTTPS (also known as web services) or JSON. Both the SOAP protocol and the JSON format are widespread and can be used from many different programming languages, consequently it is easy to implement administration scripts or create plugins for system monitoring tools such as Nagios, Munin or Cacti. Using the `xm_admin` module, NXLog can accept and initiate connections over TCP, SSL, and Unix domain sockets depending on its [configuration](#). It is advisable to prefer this module over the `xm_soapadmin` module since this module is intended as a replacement and `xm_soapadmin` will eventually be phased out.

Note that though the module can both initiate and accept connections, the direction of the HTTP requests is always the same: requests are sent to the module and it returns HTTP responses.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.1.1. Configuration

The `xm_soapadmin` module accepts the following directives in addition to the [common module directives](#).

Connect

This directive instructs the module to connect to a remote socket. The argument must be an IP address when `SocketType` is set to `TCP` or `SSL`. Otherwise it must be a name of a socket for `UDS`. `Connect` cannot be used together with the `Listen` directive. Multiple `xm_soapadmin` instances may be configured if multiple administration ports are required.

Listen

This directive instructs the module to accept connections. The argument must be an IP address when `SocketType` is `TCP` or `SSL`. Otherwise it must be the name of a socket for `UDS`. `Listen` cannot be used together with the `Connect` directive. Multiple `xm_admin` instances may be configured if multiple administration ports are required. If neither `Listen` nor `Connect` are specified, the module will listen with `SocketType TCP` on `127.0.0.1`.

Port

This specifies the port number used with the `Listen` or `Connect` modes. The default port is `8080`.

ACL

This block defines directories which can be used with the `GetFile` and `PutFile` web service requests. The name of the ACL is used in these requests together with the filename. The filename can contain only characters `[a-zA-Z0-9-_]`, so these file operations will only work within the directory.

AllowRead

If set to TRUE, `GetFile` requests are allowed.

AllowWrite

If set to TRUE, `PutFile` requests are allowed.

Directory

The name of the directory where the files are saved to or loaded from.

Example 381. ACL Block Allowing Read and Write on Files in the Directory

This ACL is named "conf" and allows both [GetFile](#) and [PutFile](#) requests on the specified directory.

```
<ACL conf>
    Directory /var/run/nxlog/configs
    AllowRead TRUE
    AllowWrite TRUE
</ACL>
```

AllowIP

This optional directive can be used to specify an IP address or a network that is allowed to connect. The directive can be specified more than once to add different IPs or networks to the whitelist. The following formats can be used:

- **0.0.0.0** (IPv4 address)
- **0.0.0.0/32** (IPv4 network with subnet bits)
- **0.0.0.0/0.0.0.0** (IPv4 network with subnet address)
- **aa::1** (IPv6 address)
- **aa::12/64** (IPv6 network with subnet bits)

AllowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with unknown and self-signed certificates. The default value is FALSE: all connections must present a trusted certificate. This directive is only valid if [SocketType](#) is set to [SSL](#).

CADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format. This directive is only valid if [SocketType](#) is set to [SSL](#).

CAFfile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote socket. This directive is only valid if [SocketType](#) is set to [SSL](#).

CertFile

This specifies the path of the certificate file to be used for SSL connections. This directive is only valid if [SocketType](#) is set to [SSL](#).

CertKeyFile

This specifies the path of the certificate key file to be used for SSL connections. This directive is only valid if [SocketType](#) is set to [SSL](#).

CRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format. This directive is only valid if [SocketType](#) is set to [SSL](#).

CRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote socket. This directive is only valid if [SocketType](#) is set to [SSL](#).

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [CertKeyFile](#). This directive is only valid if [SocketType](#) is set to [SSL](#). This directive is not needed for password-less private keys.

Reconnect

This directive has been deprecated as of version 2.4. The module will try to reconnect automatically at increasing intervals on all errors.

RequireCert

This boolean directive specifies that the remote must present a certificate. If set to TRUE and there is no certificate presented during the connection handshake, the connection will be refused. The default value is TRUE: each connection must use a certificate. This directive is only valid if [SocketType](#) is set to [SSL](#).

SocketType

This directive sets the connection type. It can be one of the following:

SSL

TLS/SSL for secure network connections

TCP

TCP, the default if [SocketType](#) is not explicitly specified

UDS

Unix domain socket

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: [SSLv2](#), [SSLv3](#), [TLSv1](#), [TLSv1.1](#), and [TLSv1.2](#). By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If [SSLProtocol](#) is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

86.1.2. Exported SOAP Methods and JSON Objects

The [xm_admin](#) module exports the following SOAP methods (web services) which can be called remotely. There is a WSDL file which can be used by different developer tools to easily hook into the exported WS API to reduce development time.

GetFile

Download a file from the NXLog agent. This will only work if the specified [ACL](#) exists.

GetLog

Download the log file from the NXLog agent.

ModuleInfo

Request information about a module instance.

ModuleRestart

Restart a module instance.

ModuleStart

Start a module instance.

ModuleStop

Stop a module instance.

PutFile

Upload a file to the NXLog agent. This will only work if the specified [ACL](#) exists. A file can be a configuration file, certificate or certificate key, pattern database, correlation rule file, etc. Using this method enables NXLog to be reconfigured from a remote host.

ServerInfo

Request information about the server. This will also return info about all module instances.

ServerRestart

Restart the server.

ServerStart

Start all modules of the server, the opposite of [ServerStop](#).

ServerStop

Stop all modules of the server. Note that the NXLog process will not exit, otherwise it would be impossible to make it come back online remotely. Extension modules are not stopped for the same reason.

The same SOAP methods were used to create an equivalent JSON Schema, so JSON Objects can be used instead of SOAP methods. This is illustrated better in the [following examples](#).

86.1.3. Request - Response Examples

This is a typical SOAP ServerInfo request and its response.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <adm:serverInfo xmlns:adm="http://log4ensics.com/2010/AdminInterface"/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <adm:serverInfoReply xmlns:adm='http://log4ensics.com/2010/AdminInterface'>
      <started>1508401312424622</started>
      <load>0.2000</load>
      <pid>15519</pid>
      <mem>12709888</mem>
      <version>3.99.2802</version>
      <os>Linux</os>
      <systeminfo>OS: Linux, Hostname: voyager, Release: 4.4.0-96-generic, Version: #119-Ubuntu SMP
Tue Sep 12 14:59:54 UTC 2017, Arch: x86_64, 4 CPU(s), 15.7Gb memory</systeminfo>
        <hostname>voyager</hostname>
        <servertime>1508405764586118</servertime>
      </adm:serverInfoReply>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

The equivalent request and response using JSON.

```
{
  "msg": {
    "command": "serverInfo"
  }
}
```

```
{
  "response": "serverInfoReply",
  "status": "success",
  "data": {
    "server-info": {
      "started": 1508401312424622,
      "load": 0.0599999865889549,
      "pid": 15519,
      "mem": 12742656,
      "os": "Linux",
      "version": "3.99.2802",
      "systeminfo": "OS: Linux, Hostname: voyager, Release: 4.4.0-96-generic, Version: #119-Ubuntu
SMP Tue Sep 12 14:59:54 UTC 2017, Arch: x86_64, 4 CPU(s), 15.7Gb memory",
      "hostname": "voyager",
      "servertime": 1508406647673758,
      "modules": {}
    }
  }
}
```

An example of a SOAP PutFile request and its response.

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <adm:putFile xmlns:adm="http://log4ensics.com/2010/AdminInterface">
      <filetype>tmp</filetype>
      <filename>test.tmp</filename>
      <file>File Content
A newline
      </file>
    </adm:putFile>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'>
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <adm:putFileReply xmlns:adm='http://log4ensics.com/2010/AdminInterface' />
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The equivalent request and response using JSON.

```
{
  "msg": {
    "command": "putFile",
    "params": {
      "filetype": "tmp",
      "filename": "test.tmp",
      "file": "File content\nA newline\n"
    }
  }
}
```

```
{
  "response": "putFileReply",
  "status": "success",
  "data": {}
}
```

86.1.4. Examples

Example 382. Configuration for Multiple Admin Ports

This configuration specifies two additional administration ports on localhost.

nxlog.conf

```
1 <Extension ssl_connect>
2   Module      xm_admin
3   Connect     192.168.1.1
4   Port        4041
5   SocketType  SSL
6   CAFile      %CERTDIR%/ca.pem
7   CertFile    %CERTDIR%/client-cert.pem
8   CertKeyFile %CERTDIR%/client-key.pem
9   KeyPass     secret
10  AllowUntrusted FALSE
11  RequireCert  TRUE
12  Reconnect    60
13  <ACL conf>
14    Directory  %CONFDIR%
15    AllowRead  TRUE
16    AllowWrite TRUE
17  </ACL>
18  <ACL cert>
19    Directory  %CERTDIR%
20    AllowRead  TRUE
21    AllowWrite TRUE
22  </ACL>
23 </Extension>
24
25 <Extension tcp_listen>
26   Module      xm_admin
27   Listen      localhost
28   Port        8080
29 </Extension>
30
31 <Extension tcp_connect>
32   Module      xm_admin
33   Connect     localhost
34   Port        4040
35 </Extension>
```

86.2. AIX Auditing (xm_aixaudit)

This module parses events in the AIX Audit format. This module is normally used in combination with the [im_file](#) module to read events from a log file. An [\[config_inputtype\]](#) is registered using the name of the module instance. See also [im_aixaudit](#), which reads audit events directly from the kernel—it is recommended instead in cases where NXLog is running as a local agent on the system.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.2.1. Configuration

The *xm_aixaudit* module accepts only the [common module directives](#).

86.2.2. Fields

The following fields are used by xm_aixaudit.

\$raw_event (type: string)

A list of event fields in key-value pairs.

\$Command (type: string)

The command executed.

\$Event (type: string)

The type of event (for example, `login`).

\$ParentPID (type: integer)

The parent process ID (PID).

\$PID (type: integer)

The process ID (PID).

\$Status (type: integer)

The status ID of the event.

\$Thread (type: integer)

The kernel thread ID, local to the process.

\$Time1 (type: datetime)

The first timestamp in the audit record.

\$Time2 (type: datetime)

The second timestamp in the audit record.

86.2.3. Examples

Example 383. Parsing AIX Audit Events

This configuration reads AIX audit logs from file and parses them with the `[config_inputtype]` registered by xm_aixaudit.

`nxlog.conf`

```
1 <Extension aixaudit_parser>
2   Module      xm_aixaudit
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        '/var/log/audit.log'
8   InputType   aixaudit_parser
9 </input>
```

86.3. Apple System Logs (xm_asl)

This module provides support for parsing Apple System Log (ASL) files. It registers an `InputType` using the name of the module instance. This module can be used with the `im_file` module.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.3.1. Configuration

The `xm_asl` module accepts only the [common module directives](#).

86.3.2. Fields

The following fields are used by `xm_asl`.

`$raw_event` (type: `string`)

The raw log message.

`$EventTime` (type: `datetime`)

A timestamp for when the event was created by the ASL daemon.

`$Facility` (type: `string`)

The sender's facility.

`$GroupAccess` (type: `integer`)

The GID of the group that has permission to read the message (-1 for "all groups").

`$Level` (type: `string`)

The ASL record level string. See `$Severity`.

`$LevelValue` (type: `integer`)

The ASL record level value corresponding to the `$Level`.

`$RecordId` (type: `integer`)

A numeric ID for this record.

`$Sender` (type: `string`)

The name of the process that sent the message.

`$SenderGid` (type: `integer`)

The group ID (GID) of the process that generated the event (-1 or -2 may indicate the `nobody` or `nogroup` groups; see [/etc/group](#) on the source system).

`$SenderHost` (type: `string`)

The host that the sender belongs to (usually the name of the device).

`$SenderPid` (type: `integer`)

The ID of the process that generated the event.

`$SenderUid` (type: `integer`)

The user ID (UID) of the process that generated the event (-2 may indicate the `nobody` group; see [/etc/group](#) on the source system).

`$Severity` (type: `string`)

The normalized severity of the event, mapped as follows.

ASL Level	Normalized Severity
0/EMERGENCY	5/CRITICAL
1/ALERT	5/CRITICAL
2/CRITICAL	5/CRITICAL
3/ERROR	4/ERROR
4/WARNING	3/WARNING
5/NOTICE	2/INFO
6/INFO	2/INFO
7/DEBUG	1/DEBUG

\$SeverityValue (type: *integer*)

The normalized severity number of the event. See [\\$Severity](#).

\$UserAccess (type: *integer*)

The UID of the user that has permission to read the message (-1 for "all users").

86.3.3. Examples

Example 384. Parsing Apple System Logs With xm_asl

This example uses an [im_file](#) module instance to read an ASL log file and the [InputType](#) provided by *xm_asl* to parse the events. The various [Fields](#) are added to the event record.

nxlog.conf

```

1 <Extension asl_parser>
2   Module      xm_asl
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        "tmp/input.asl"
8   InputType   asl_parser
9 </Input>
```

86.4. Basic Security Module Auditing (xm_bsm)

This module provides support for parsing events logged to file using Sun's Basic Security Module (BSM) Auditing API. This module is normally used in combination with the [im_file](#) module to read events from a log file. An [\[config_inputtype\]](#) is registered using the name of the module instance. See also [im_bsm](#), which reads audit events directly from the kernel—it is recommended instead in cases where NXLog is running as a local agent on the system.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.4.1. Configuration

The *xm_bsm* module accepts only the [common module directives](#).

86.4.2. Fields

The following fields are used by xm_bsm.

\$Arbitrary (type: *string*)

Arbitrary data token associated with the event, if any.

\$Arg0.Description (type: *string*)

The description of argument 0.

\$Arg0.Value (type: *string*)

The value of argument 0.

\$AttributeDevID (type: *string*)

The file device ID.

\$AttributeFsID (type: *string*)

The file system ID.

\$AttributeGID (type: *string*)

The file owner group ID (GID).

\$AttributeMode (type: *string*)

The file access mode.

\$AttributeNodeID (type: *string*)

The file node ID.

\$AttributeUID (type: *string*)

The file owner user ID (UID).

\$Cmd (type: *string*)

The command, with arguments and environment.

\$EventHost (type: *string*)

The event host.

\$EventModifier (type: *string*)

The event modifier.

\$EventTime (type: *datetime*)

The time when the event began.

\$EventType (type: *string*)

The type of the event.

\$ExecArgs (type: *string*)

The list of arguments.

\$ExecEnv (type: *string*)

The list of environment variables.

\$ExitErrno (type: *string*)

The exit status.

\$ExitRetval (type: *string*)

The exit return value.

\$FileModificationTime (type: *datetime*)

The last modified time of the file corresponding to the event (if applicable).

\$FileName (type: *string*)

The name of the file corresponding to the event (if applicable).

\$Hostname (type: *string*)

The IP address or hostname where the event originated.

\$IPAddress (type: *string*)

The IP address.

\$IPC (type: *string*)

The IPC token.

\$IPChecksum (type: *string*)

The checksum in the IP header.

\$IPCPPermCreatorGID (type: *string*)

The IPC creator group ID (GID).

\$IPCPPermCreatorUID (type: *string*)

The IPC creator user ID (UID).

\$IPCPPermGID (type: *string*)

The IPC owner group ID (GID).

\$IPCPPermKey (type: *string*)

The IPC permission key.

\$IPCPPermMode (type: *string*)

The IPC access mode.

\$IPCPPermSeqID (type: *string*)

The IPC slot sequence.

\$IPCPPermUID (type: *string*)

The IPC owner user ID (UID).

\$IPDestAddr (type: *string*)

The destination address in the IP header.

\$IPFragmentOffset (type: *string*)

The fragment offset field of the IP header.

\$IPHeaderLen (type: *string*)

The total length of the IP header.

\$IPIIdent (type: *string*)

The ID of the IP header.

\$IPProto (type: *string*)

The IP protocol.

\$IPServiceType (type: *string*)

The IP type of service (TOS).

\$IPSrcAddr (type: *string*)

The source address in the IP header.

\$IPTTL (type: *string*)

The time-to-live (TTL) of the IP header.

\$IPVer (type: *string*)

The IP version.

\$Opaque (type: *string*)

The opaque field (hexadecimal).

\$Path (type: *string*)

The object path token.

\$Privilege (type: *string*)

The privilege token.

\$ProcessAuditID (type: *string*)

The audit ID in the Process section.

\$ProcessGID (type: *string*)

The effective group ID (GID) in the Process section.

\$ProcessPID (type: *string*)

The process ID (PID) in the Process section.

\$ProcessRealGID (type: *string*)

The real group ID (GID) in the Process section.

\$ProcessRealUID (type: *string*)

The real user ID (UID) in the Process section.

\$ProcessSID (type: *string*)

The session ID (SID) in the Process section.

\$ProcessTerminal.Host (type: *string*)

The terminal IP address in the Process section.

\$ProcessTerminal.Port (type: *string*)

The terminal port in the Process section.

\$ProcessUID (type: *string*)

The effective user ID (UID) in the Process section.

\$ReturnErrno (type: *string*)

The error status in the Return section.

\$ReturnRetval (type: *string*)

The return value in the Return section.

\$Sequence (type: *string*)

The sequence number.

\$SocketAddress (type: *string*)

The socket address.

\$SocketPort (type: *string*)

The socket port.

\$SocketType (type: *string*)

The socket type.

\$SubjectAuditID (type: *string*)

The audit ID in the Subject section.

\$SubjectGID (type: *string*)

The effective group ID (GID) in the Subject section.

\$SubjectPID (type: *string*)

The process ID (PID) in the Subject section.

\$SubjectRealGID (type: *string*)

The real group ID (GID) in the Subject section.

\$SubjectRealUID (type: *string*)

The real user ID (UID) in the Subject section.

\$SubjectSID (type: *string*)

The session ID (SID) in the Subject section.

\$SubjectTerminal.Host (type: *string*)

The terminal IP address in the Subject section.

\$SubjectTerminal.Port (type: *string*)

The terminal port in the Subject section.

\$SubjectUID (type: *string*)

The effective user ID (UID) in the Subject section.

\$TerminalAddress (type: *string*)

Terminal address.

\$TerminalLocalPort (type: *string*)

The terminal local port.

\$TerminalRemotePort (type: *string*)

The terminal remote port.

\$Text (type: *string*)

A text string associated with the event.

\$TokenVersion (type: *string*)

The token version.

\$Zone (type: *string*)

The zone name.

86.4.3. Examples

Example 385. Parsing BSM Events With xm_bsm

This configuration reads BSM audit logs from file and parses them with the `InputType` registered by `xm_bsm`.

nxlog.conf

```
1 <Extension bsm_parser>
2   Module      xm_bsm
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        '/var/audit/*'
8   InputType   bsm_parser
9 </input>
```

86.5. CEF (xm_cef)

This module provides functions for generating and parsing data in the Common Event Format (CEF) used by HP ArcSight™ products. For more information about the format see the [ArcSight Common Event Format \(CEF\) Guide](#).

NOTE

CEF uses Syslog as a transport. For this reason the `xm_syslog` module must be used in conjunction with `xm_cef` in order to parse or generate the additional Syslog header, unless the CEF data is used without Syslog. See [examples for both cases](#) below.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.5.1. Configuration

The `xm_cef` module accepts only the [common module directives](#).

86.5.2. Functions

The following functions are exported by `xm_cef`.

***string* `to_cef()`**

Convert the specified fields to a single CEF formatted string.

86.5.3. Procedures

The following procedures are exported by `xm_cef`.

```
parse_cef();
```

Parse the `$raw_event` field as CEF input.

```
parse_cef(string source);
```

Parse the given string as CEF format.

```
to_cef();
```

Format the specified fields as CEF and put this into the `$raw_event` field. The header fields can be overridden by values contained in `$CEFVersion`, `$CEFDDeviceVendor`, `$CEFDDeviceProduct`, `$CEFDDeviceVersion`, `$CEFSignatureID`, `$CEFName`, and `$CEFSeverity`.

86.5.4. Examples

Example 386. Sending Windows EventLog as CEF over UDP

This configuration collects both Windows EventLog and NXLog internal messages, converts to CEF with Syslog headers, and forwards via UDP.

nxlog.conf

```
1 <Extension cef>
2     Module xm_cef
3 </Extension>
4
5 <Extension syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input internal>
10    Module im_internal
11 </Input>
12
13 <Input eventlog>
14    Module im_msvistalog
15 </Input>
16
17 <Output udp>
18    Module om_udp
19    Host    192.168.168.2
20    Port    1514
21    Exec    $Message = to_cef(); to_syslog_bsd();
22 </Output>
23
24 <Route arcsight>
25    Path    internal, eventlog => udp
26 </Route>
```

Example 387. Parsing CEF

The following configuration receives CEF over UDP and converts the parsed data into JSON.

nxlog.conf

```

1 <Extension cef>
2   Module xm_cef
3 </Extension>
4
5 <Extension syslog>
6   Module xm_syslog
7 </Extension>
8
9 <Extension json>
10  Module xm_json
11 </Extension>
12
13 <Input udp>
14   Module im_udp
15   Host 0.0.0.0
16   Exec parse_syslog(); parse_cef($Message);
17 </Input>
18
19 <Output file>
20   Module om_file
21   File "cef2json.log"
22   Exec to_json();
23 </Output>
24
25 <Route cef2json>
26   Path udp => file
27 </Route>
```

86.6. Character Set Conversion (xm_charconv)

This module provides tools for converting strings between different character sets (codepages). All the encodings available to *iconv* are supported. See [iconv -1](#) for a list of encoding names.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.6.1. Configuration

The *xm_charconv* module accepts the following directives in addition to the [common module directives](#).

AutodetectCharsets

This optional directive accepts a comma-separated list of character set names. When **auto** is specified as the source encoding for [convert\(\)](#) or [convert_fields\(\)](#), these character sets will be tried for conversion. This directive is not related to the [LineReader](#) directive or the corresponding [InputType](#) registered by the module.

BigEndian

This optional boolean directive specifies the endianness to use during the encoding conversion. If this directive is not specified, it defaults to the host's endianness. This directive only affects the registered [InputType](#) and is only applicable if the [LineReader](#) directive is set to a non-Unicode encoding and [CharBytes](#) is set to 2 or 4.

CharBytes

This optional integer directive specifies the byte-width of the encoding to use during conversion. Acceptable values are 1 (the default), 2, and 4. Most variable width encodings will work with the default value. This directive only affects the registered [InputType](#) and is only applicable if the [LineReader](#) directive is set to a non-Unicode encoding.

LineReader

If this optional directive is specified with an encoding, an [InputType](#) will be registered using the name of the *xm_charconv* module instance. The following Unicode encodings are supported: UTF-8, UCS-2, UCS-2BE, UCS-2LE, UCS-4, UCS-4BE, UCS-4LE, UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE, and UTF-7. For other encodings, it may be necessary to also set [BigEndian](#) and/or [CharBytes](#).

86.6.2. Functions

The following functions are exported by xm_charconv.

```
string convert(string source, string srcencoding, string dstencoding)
```

Convert the *source* string to the encoding specified in *dstencoding* from *srcencoding*. The *srcencoding* argument can be set to **auto** to request auto detection.

86.6.3. Procedures

The following procedures are exported by xm_charconv.

```
convert_fields(string srcencoding, string dstencoding);
```

Convert all string type fields of a log message from *srcencoding* to *dstencoding*. The *srcencoding* argument can be set to **auto** to request auto detection.

86.6.4. Examples

Example 388. Character set auto-detection of various input encodings

This configuration shows an example of character set auto-detection. The input file can contain differently encoded lines, and the module normalizes output to UTF-8.

nxlog.conf

```
1 <Extension charconv>
2   Module           xm_charconv
3   AutodetectCharsets utf-8, euc-jp, utf-16, utf-32, iso8859-2
4 </Extension>
5
6 <Input filein>
7   Module           im_file
8   File             "tmp/input"
9   Exec             convert_fields("auto", "utf-8");
10 </Input>
11
12 <Output fileout>
13   Module           om_file
14   File             "tmp/output"
15 </Output>
16
17 <Route r>
18   Path             filein => fileout
19 </Route>
```

Example 389. Registering and Using an InputType

This configuration uses the [InputType](#) registered via the [LineReader](#) directive to read a file with the ISO-8859-2 encoding.

nxlog.conf

```
1 <Extension charconv>
2   Module      xm_charconv
3   LineReader  ISO-8859-2
4 </Extension>
5
6 <Input in>
7   Module      im_file
8   File        'modules/extension/charconv/iso-8859-2.in'
9   InputType   charconv
10 </Input>
```

86.7. CSV (xm_csv)

This module provides functions and procedures for working with data formatted as comma-separated values (CSV). CSV input can be parsed into [fields](#) and CSV output can be generated.

The [pm_transformer](#) module provides a simple interface to parse and generate CSV format, but the *xm_csv* module exports an API that can be used to solve more complex tasks involving CSV formatted data.

NOTE

It is possible to use more than one *xm_csv* module instance with different options in order to support different CSV formats at the same time. For this reason, functions and procedures exported by the module are public and must be referenced by the module instance name.

See the list of installer packages that provide this module in the [NXLog User Guide](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.7.1. Configuration

The *xm_csv* module accepts the following directives in addition to the [common module directives](#). The [Fields](#) directive is required.

Fields

This mandatory directive accepts a comma-separated list of fields which will be filled from the input parsed. Field names with or without the dollar sign (\$) are accepted. The fields will be stored as [strings](#) unless their types are explicitly specified with the [FieldTypes](#) directive.

Delimiter

This optional directive takes a single character (see [below](#)) as argument to specify the delimiter character used to separate fields. The default delimiter character is the comma (,). Note that there is no delimiter after the last field.

EscapeChar

This optional directive takes a single character (see [below](#)) as argument to specify the escape character used to escape special characters. The escape character is used to prefix the following characters: the escape character itself, the [quote character](#), and the [delimiter character](#). If [EscapeControl](#) is TRUE, the newline (\n), carriage return (\r), tab (\t), and backspace (\b) control characters are also escaped. The default escape

character is the backslash character (\).

EscapeControl

If this optional boolean directive is set to TRUE, control characters are also escaped. See the [EscapeChar](#) directive for details. The default is TRUE: control characters are escaped. Note that this is necessary to allow single line CSV field lists which contain line-breaks.

FieldTypes

This optional directive specifies the list of types corresponding to the field names defined in [Fields](#). If specified, the number of types must match the number of field names specified with [Fields](#). If this directive is omitted, all fields will be stored as [strings](#). This directive has no effect on the fields-to-CSV conversion.

QuoteChar

This optional directive takes a single character (see [below](#)) as argument to specify the quote character used to enclose fields. If [QuoteOptional](#) is TRUE, then only [string](#) type fields are quoted. The default is the double-quote character (").

QuoteMethod

This optional directive can take the following values:

All

All fields will be quoted.

None

Nothing will be quoted. This can be problematic if a field value (typically text that can contain any character) contains the delimiter character. Make sure that this is escaped or replaced with something else.

String

Only [string](#) type fields will be quoted. This has the same effect as [QuoteOptional](#) set to TRUE and is the default behavior if the [QuoteMethod](#) directive is not specified.

Note that this directive only effects CSV generation when using [to_csv\(\)](#). The CSV parser can automatically detect the quotation.

QuoteOptional

This directive has been deprecated in favor of [QuoteMethod](#), which should be used instead.

StrictMode

If this optional boolean directive is set to TRUE, the CSV parser will fail to parse CSV lines that do not contain the required number of fields. When this is set to FALSE and the input contains fewer fields than specified in [Field](#), the rest of the fields will be unset. The default value is FALSE.

UndeValue

This optional directive specifies a string which will be treated as an undefined value. This is particularly useful when parsing the W3C format where the dash (-) marks an omitted field.

86.7.1.1. Specifying Quote, Escape, and Delimiter Characters

The [QuoteChar](#), [EscapeChar](#), and [Delimiter](#) directives can be specified in several ways.

Unquoted single character

Any printable character can be specified as an unquoted character, except for the backslash (\):

Delimiter ;

Control characters

The following non-printable characters can be specified with escape sequences:

\a
audible alert (bell)

\b
backspace

\t
horizontal tab

\n
newline

\v
vertical tab

\f
formfeed

\r
carriage return

For example, to use TAB delimiting:

```
Delimiter \t
```

A character in single quotes

The configuration parser strips whitespace, so it is not possible to define a space as the delimiter unless it is enclosed within quotes:

```
Delimiter ' '
```

Printable characters can also be enclosed:

```
Delimiter ';'
```

The backslash can be specified when enclosed within quotes:

```
Delimiter '\\'
```

A character in double quotes

Double quotes can be used like single quotes:

```
Delimiter " "
```

The backslash can be specified when enclosed within double quotes:

```
Delimiter "\\"
```

A hexadecimal ASCII code

Hexadecimal ASCII character codes can also be used by prepending **0x**. For example, the space can be specified as:

```
Delimiter 0x20
```

This is equivalent to:

```
Delimiter " "
```

86.7.2. Functions

The following functions are exported by xm_csv.

`string to_csv()`

Convert the specified fields to a single CSV formatted string.

86.7.3. Procedures

The following procedures are exported by xm_csv.

`parse_csv();`

Parse the `$raw_event` field as CSV input.

`parse_csv(string source);`

Parse the given string as CSV format.

`to_csv();`

Format the specified fields as CSV and put this into the `$raw_event` field.

86.7.4. Examples

Example 390. Complex CSV Format Conversion

This example shows that the *xm_csv* module can not only parse and create CSV formatted input and output, but with multiple *xm_csv* modules it is also possible to reorder, add, remove, or modify fields before outputting to a different CSV format.

nxlog.conf

```

1 <Extension csv1>
2   Module      xm_csv
3   Fields      $id, $name, $number
4   FieldTypes  integer, string, integer
5   Delimiter   ,
6 </Extension>
7
8 <Extension csv2>
9   Module      xm_csv
10  Fields      $id, $number, $name, $date
11  Delimiter   ;
12 </Extension>
13
14 <Input in>
15   Module      im_file
16   File        "tmp/input"
17   <Exec>
18     csv1->parse_csv();
19     $date = now();
20     if not defined $number $number = 0;
21     csv2->to_csv();
22   </Exec>
23 </Input>
24
25 <Output out>
26   Module      om_file
27   File        "tmp/output"
28 </Output>
```

Input Sample

```

1, "John K.", 42
2, "Joe F.", 43
```

Output Sample

```

1;42;"John K.";2011-01-15 23:45:20
2;43;"Joe F.";2011-01-15 23:45:20
```

86.8. External Program Execution (*xm_exec*)

This module provides two procedures which make it possible to execute external scripts or programs. These two procedures are provided through this extension module in order to keep the NXLog core small. Also, without this module loaded an administrator is not able to execute arbitrary scripts.

NOTE

The *im_exec* and *om_exec* modules also provide support for running external programs, though the purpose of these is to pipe data to and read data from programs. The procedures provided by the *xm_exec* module do not pipe log message data, but are intended for multiple invocations (though data can be still passed to the executed script/program as command line arguments).

86.8.1. Configuration

The `xm_exec` module accepts only the [common module directives](#).

86.8.2. Procedures

The following procedures are exported by `xm_exec`.

```
exec(string command, varargs args);
```

Execute `command`, passing it the supplied arguments, and wait for it to terminate. The command is executed in the caller module's context. Note that the module calling this procedure will block until the process terminates. Use the `exec_async()` procedure to avoid this problem. All output written to standard output and standard error by the spawned process is discarded.

```
exec_async(string command, varargs args);
```

This procedure executes the command passing it the supplied arguments and does not wait for it to terminate.

86.8.3. Examples

Example 391. NXLog Acting as a Cron Daemon

This `xm_exec` module instance will run the command every second without waiting for it to terminate.

nxlog.conf

```
1 <Extension exec>
2   Module xm_exec
3   <Schedule>
4     Every 1 sec
5     Exec   exec_async("/bin/true");
6   </Schedule>
7 </Extension>
```

Example 392. Sending Email Alerts

If the `$raw_event` field matches the regular expression, an email will be sent.

`nxlog.conf`

```
1 <Extension exec>
2   Module xm_exec
3 </Extension>
4
5 <Input tcp>
6   Module im_tcp
7   Host 0.0.0.0
8   Port 1514
9   <Exec>
10     if $raw_event =~ /alertcondition/
11     {
12       exec_async("/bin/sh", "-c", 'echo "' + $Hostname +
13 '\n\nRawEvent:\n' + $raw_event +
14 '"|/usr/bin/mail -a "Content-Type: text/plain; charset=UTF-8" -s "ALERT" ' +
15 'user@domain.com');
16     }
17   </Exec>
18 </Input>
19
20 <Output file>
21   Module om_file
22   File   "/var/log/messages"
23 </Output>
24
25 <Route tcp_to_file>
26   Path   tcp => file
27 </Route>
```

For another example, see [File Rotation Based on Size](#).

86.9. Filelist (xm_filelist)

The `xm_filelist` module can be used to implement file-based blacklisting or whitelisting. This extension module provides two functions, [contains\(\)](#) and [matches\(\)](#), that can be invoked to check whether the string argument is present in the file. This can be a username, IP address, or similar. Each referenced file is cached in memory and any modifications are automatically loaded without the need to reload NXLog.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.9.1. Configuration

The `xm_filelist` module accepts the following directives in addition to the [common module directives](#). The `File` directive is required.

`File`

The mandatory `File` directive specifies the name of the file that will be read into memory. This directive may be specified more than once if multiple files need to be operated on.

CheckInterval

This optional directive specifies the frequency with which the files are checked for modifications, in seconds. The default value is 5 seconds. File checks are disabled if **CheckInterval** is set to 0.

86.9.2. Functions

The following functions are exported by xm_filelist.

`boolean contains(string str)`

Check if line in the file(s) contains the string *str*.

`boolean contains(string str, boolean caseinsensitive)`

Check if line in the file(s) contains the string *str*. May be case insensitive according to *caseinsensitive*.

`boolean matches(string str)`

Check if a line in the file(s) matches the string *str*.

`boolean matches(string str, boolean caseinsensitive)`

Check if a line in the file(s) matches the string *str*. May be case insensitive according to *caseinsensitive*.

86.10. File Operations (xm_fileop)

This module provides functions and procedures to manipulate files. Coupled with a **Schedule** block, this module allows various log rotation and retention policies to be implemented, including:

- log file retention based on file size,
- log file retention based on file age, and
- cyclic log file rotation and retention.

NOTE

Rotating, renaming, or removing the file written by `om_file` is also supported with the help of the `om_file reopen()` procedure.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.10.1. Configuration

The `xm_fileop` module accepts only the [common module directives](#).

86.10.2. Functions

The following functions are exported by xm_fileop.

`boolean dir_exists(string path)`

Return TRUE if *path* exists and is a directory. On error undef is returned and an error is logged.

`string dir_temp_get()`

Return the name of a directory suitable as a temporary storage location.

`string file_basename(string file)`

Strip the directory name from the full *file* path. For example, `basename(' /var/log/app.log')` will return `app.log`.

datetime file_ctime(string file)

Return the creation or inode-changed time of *file*. On error undef is returned and an error is logged.

string file dirname(string file)

Return the directory name of the full *file* path. For example, `basename('/var/log/app.log')` will return `/var/log`. Returns an empty string if *file* does not contain any directory separators.

boolean file_exists(string file)

Return TRUE if *file* exists and is a regular file.

integer file_inode(string file)

Return the inode number of *file*. On error undef is returned and an error is logged.

datetime file_mtime(string file)

Return the last modification time of *file*. On error undef is returned and an error is logged.

string file_read(string file)

Return the contents of *file* as a string value. On error undef is returned and an error is logged.

integer file_size(string file)

Return the size of *file*, in bytes. On error undef is returned and an error is logged.

string file_type(string file)

Return the type of *file*. The following string values can be returned: FILE, DIR, CHAR, BLOCK, PIPE, LINK, SOCKET, and UNKNOWN. On error undef is returned and an error is logged.

86.10.3. Procedures

The following procedures are exported by xm_fileop.

dir_make(string path);

Create a directory recursively (like `mkdir -p`). It succeeds if the directory already exists. An error is logged if the operation fails.

dir_remove(string file);

Remove the directory from the filesystem.

file_append(string src, string dst);

Append the contents of the file *src* to *dst*. The *dst* file will be created if it does not exist. An error is logged if the operation fails.

file_chmod(string file, integer mode);

Change the permissions of *file*. This function is only implemented on POSIX systems where chmod() is available in the underlying operating system. An error is logged if the operation fails.

file_chown(string file, integer uid, integer gid);

Change the ownership of *file*. This function is only implemented on POSIX systems where chown() is available in the underlying operating system. An error is logged if the operation fails.

file_chown(string file, string user, string group);

Change the ownership of *file*. This function is only implemented on POSIX systems where chown() is available in the underlying operating system. An error is logged if the operation fails.

```
file_copy(string src, string dst);
```

Copy the file *src* to *dst*. If file *dst* already exists, its contents will be overwritten. An error is logged if the operation fails.

```
file_cycle(string file);
```

Do a cyclic rotation on *file*. The file will be moved to "*file.1*". If "*file.1*" already exists it will be moved to "*file.2*", and so on. Wildcards are supported in the *file* path and filename. The backslash (\) must be escaped if used as the directory separator with wildcards (for example, C:\\test*.log). This procedure will reopen the LogFile if it is cycled. An error is logged if the operation fails.

```
file_cycle(string file, integer max);
```

Do a cyclic rotation on *file* as described above. The *max* argument specifies the maximum number of files to keep. For example, if *max* is 5, "*file.6*" will be deleted.

```
file_link(string src, string dst);
```

Create a hardlink from *src* to *dst*. An error is logged if the operation fails.

```
file_remove(string file);
```

Remove *file*. It is possible to specify a wildcard in the filename (but not in the path). The backslash (\) must be escaped if used as the directory separator with wildcards (for example, C:\\test*.log). This procedure will reopen the LogFile if it is removed. An error is logged if the operation fails.

```
file_remove(string file, datetime older);
```

Remove *file* if its creation time is older than the value specified in *older*. It is possible to specify a wildcard in the filename (but not in the path). The backslash (\) must be escaped if used as the directory separator with wildcards (for example, C:\\test*.log). This procedure will reopen the LogFile if it is removed. An error is logged if the operation fails.

```
file_rename(string old, string new);
```

Rename the file *old* to *new*. If the file *new* exists, it will be overwritten. Moving files or directories across devices may not be possible. This procedure will reopen the LogFile if it is renamed. An error is logged if the operation fails.

```
file_touch(string file);
```

Update the last modification time of *file* or create the *file* if it does not exist. An error is logged if the operation fails.

```
file_truncate(string file);
```

Truncate *file* to zero length. If the *file* does not exist, it will be created. An error is logged if the operation fails.

```
file_truncate(string file, integer offset);
```

Truncate *file* to the size specified in *offset*. If the *file* does not exist, it will be created. An error is logged if the operation fails.

```
file_write(string file, string value);
```

Write *value* into *file*. The *file* will be created if it does not exist. An error is logged if the operation fails.

86.10.4. Examples

Example 393. Rotation of the Internal LogFile

In this example, the internal log file is rotated based on time and size.

nxlog.conf

```

1 #define LOGFILE C:\Program Files (x86)\nxlog\data\nxlog.log
2 define LOGFILE /var/log/nxlog/nxlog.log
3
4 <Extension fileop>
5   Module      xm_fileop
6
7   # Check the log file size every hour and rotate if larger than 1 MB
8   <Schedule>
9     Every    1 hour
10    Exec      if (file_size('%LOGFILE%') >= 1M) \
11          file_cycle('%LOGFILE%', 2);
12  </Schedule>
13
14  # Rotate log file every week on Sunday at midnight
15  <Schedule>
16    When      @weekly
17    Exec      file_cycle('%LOGFILE%', 2);
18  </Schedule>
19 </Extension>
```

86.11. GELF (xm_gelf)

This module provides an output writer function which can be used to generate output in Graylog Extended Log Format (GELF) for [Graylog2](#) or GELF compliant tools.

Unlike Syslog format (with Snare Agent, for example), the GELF format contains structured data in JSON so that the fields are available for analysis. This is especially convenient with sources such as the Windows EventLog which already generate logs in a structured format.

The *xm_gelf* module provides the following output writer functions:

OutputType GELF_TCP

This output writer generates GELF for use with TCP (use with the [om_tcp](#) output module).

OutputType GELF_UDP

This output writer generates GELF for use with UDP (use with the [om_udp](#) output module).

OutputType GELF

This type is equivalent to [GELF_UDP](#).

The [GELF](#) output generated by this module includes all fields, except for the \$raw_event field and any field having a leading dot (.) or underscore (_).

Configure NXLog to output GELF formatted data by following these steps:

1. Load the *xm_gelf* module:

```

1 <Extension _gelf>
2   Module      xm_gelf
3 </Extension>
```

2. Set the [OutputType](#) to [GELF_UDP](#) in the [om_udp](#) output module:

```
1 <Output out_udp>
2   Module      om_udp
3   Host       127.0.0.1
4   Port       12201
5   OutputType GELF_UDP
6 </Output>
```

Or, for `om_tcp`, use `GELF_TCP`:

```
1 <Output out_tcp>
2   Module      om_tcp
3   Host       127.0.0.1
4   Port       12201
5   OutputType GELF_TCP
6 </Output>
```

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.11.1. Configuration

The `xm_gelf` module accepts the following directives in addition to the [common module directives](#).

ShortMessageLength

This optional directive can be used to specify the length of the `short_message` field. This defaults to 64 if the directive is not explicitly specified. If the field `short_message` or `ShortMessage` is present, it will not be truncated.

UseNullDelimiter

If this optional boolean directive is TRUE, `GELF_TCP` will use the NUL delimiter. If this directive is FALSE, it will use the newline delimiter. The default is TRUE.

86.11.2. Examples

Example 394. Sending Windows EventLog to Graylog2 in GELF

The following configuration reads the Windows EventLog and sends it to a Graylog2 server in GELF format.

nxlog.conf

```
1 <Extension gelf>
2   Module      xm_gelf
3 </Extension>
4
5 <Input eventlog>
6   # Use 'im_mseventlog' for Windows XP, 2000 and 2003
7   Module      im_msvisalog
8   # Uncomment the following to collect specific event logs only
9   # but make sure not to leave any `#` as only <!-- --> style comments
10  # are supported inside the XML.
11  #Query  <QueryList> \
12  #          <Query Id="0"> \
13  #              <Select Path="Application">*</Select> \
14  #              <Select Path="System">*</Select> \
15  #              <Select Path="Security">*</Select> \
16  #          </Query> \
17  #      </QueryList>
18 </Input>
19
20 <Output udp>
21   Module      om_udp
22   Host        192.168.1.1
23   Port        12201
24   OutputType  GELF
25 </Output>
26
27 <Route eventlog_to_udp>
28   Path        eventlog => udp
29 </Route>
```

Example 395. Forwarding Custom Log Files to Graylog2 in GELF

In this example, custom application logs are collected and sent out in GELF, with custom fields set to make the data more useful for the receiver.

nxlog.conf

```
1 <Extension gelf>
2   Module      xm_gelf
3 </Extension>
4
5 <Input file>
6   Module      im_file
7   File        "/var/log/app*.log"
8
9   <Exec>
10    # Set the $EventTime field usually found in the logs by
11    # extracting it with a regexp. If this is not set, the current
12    # system time will be used which might be a little off.
13    if $raw_event =~ /(\d\d\d\d-\d\d-\d\d \d\d:\d\d:\d\d)/
14      $EventTime = parsedate($1);
15
16    # Explicitly set the Hostname. This defaults to the system's
17    # hostname if unset.
18    $Hostname = 'myhost';
19
20    # Now set the severity level to something custom. This defaults
21    # to 'INFO' if unset. We can use the following numeric values
22    # here which are the standard Syslog values: ALERT: 1, CRITICAL:
23    # 2, ERROR: 3, WARNING: 4, NOTICE: 5, INFO: 6, DEBUG: 7
24    if $raw_event =~ /ERROR/ $SyslogSeverityValue = 3;
25    else $SyslogSeverityValue = 6;
26
27    # Set a field to contain the name of the source file
28    $FileName = file_name();
29
30    # To set a custom message, use the $Message field. The
31    # $raw_event field is used if $Message is unset.
32    if $raw_event =~ /something important/
33      $Message = 'IMPORTANT!! ' + $raw_event;
34  </Exec>
35 </Input>
36
37 <Output udp>
38   Module      om_udp
39   Host        192.168.1.1
40   Port        12201
41   OutputType  GELF
42 </Output>
43
44 <Route file_to_gelf>
45   Path        file => udp
46 </Route>
```

Example 396. Parsing a CSV File and Sending it to Graylog2 in GELF

With this configuration, NXLog will read a CSV file containing three fields and forward the data in GELF so that the fields will be available on the server.

nxlog.conf

```

1 <Extension gelf>
2   Module      xm_gelf
3 </Extension>
4
5 <Extension csv>
6   Module      xm_csv
7   Fields      $name, $number, $location
8   FieldTypes  string, integer, string
9   Delimiter   ,
10 </Extension>
11
12 <Input file>
13   Module     im_file
14   File       "/var/log/app/csv.log"
15   Exec       csv->parse_csv();
16 </Input>
17
18 <Output udp>
19   Module     om_udp
20   Host       192.168.1.1
21   Port       12201
22   OutputType GELF
23 </Output>
24
25 <Route csv_to_gelf>
26   Path       file => udp
27 </Route>
```

86.12. Grok (xm_grok)

This module supports parsing events with Grok patterns. A field is added to the event record for each pattern semantic. For more information about Grok, see the [Logstash Grok filter plugin](#) documentation.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.12.1. Configuration

The *xm_grok* module accepts the following directives in addition to the [common module directives](#).

Pattern

This mandatory directive specifies a directory or file containing Grok patterns. Wildcards may be used to specify multiple directories or files. This directive may be used more than once.

86.12.2. Functions

The following functions are exported by *xm_grok*.

boolean match_grok(string pattern)

Execute the [*match_grok\(\)*](#) procedure with the specified *pattern* on the **\$raw_event** field. If the event is

successfully matched, return TRUE, otherwise FALSE.

`boolean match_grok(string field, string pattern)`

Execute the `match_grok()` procedure with the specified *pattern* on the specified *field*. If the event is successfully matched, return TRUE, otherwise FALSE.

86.12.3. Procedures

The following procedures are exported by xm_grok.

`match_grok(string pattern);`

Attempt to match and parse the `$raw_event` field of the current event with the specified *pattern*.

`match_grok(string field, string pattern);`

Attempt to match and parse the *field* of the current event with the specified *pattern*.

86.12.4. Examples

Example 397. Using Grok Patterns for Parsing

This configuration reads Syslog events from file and parses them with the `parse_syslog()` procedure (this sets the `$Message` field). Then the `match_grok()` function is used to attempt a series of matches on the `$Message` field until one is successful. If no patterns match, an internal message is logged.

`nxlog.conf`

```

1 <Extension _syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Extension grok>
6   Module xm_grok
7   Pattern modules/extension/grok/patterns2.txt
8 </Extension>
9
10 <Input in>
11   Module im_file
12   File   'test2.log'
13   <Exec>
14     parse_syslog();
15     if match_grok($Message, "%{SSH_AUTHFAIL_WRONGUSER}") {}
16     else if match_grok($Message, "%{SSH_AUTHFAIL_WRONGCREDNS}") {}
17     else if match_grok($Message, "%{SSH_AUTH_SUCCESS}") {}
18     else if match_grok($Message, "%{SSH_DISCONNECT}") {}
19     else
20     {
21       log_info('Event did not match any pattern');
22     }
23   </Exec>
24 </Input>
```

`patterns2.txt`

```

USERNAME [a-zA-Z0-9_-]+
INT(?:[+-]?(?:[0-9]+))
BASE10NUM(?:<![0-9.+-]>)(?:[+-]?(?:(?:[0-9]+(?:\.[0-9]+)?))|(?:\.[0-9]+)))
NUMBER(?:%{BASE10NUM})
WORD \b\w+\b
GREEDYDATA .*
IP(?:<![0-9]>)(?:(:?25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2})[.](?:25[0-5]|2[0-4][0-9]|0-1)?[0-9]{1,2})(?!([0-9]))(?!([0-9]))
SSH_AUTHFAIL_WRONGUSER Failed %{WORD:ssh_authmethod} for invalid user %{USERNAME:ssh_user} from
%{IP:ssh_client_ip} port %{NUMBER:ssh_client_port} (?<ssh_protocol>\w+\d+)
SSH_AUTHFAIL_WRONGCREDNS Failed %{WORD:ssh_authmethod} for %{USERNAME:ssh_user} from
%{IP:ssh_client_ip} port %{NUMBER:ssh_client_port} (?<ssh_protocol>\w+\d+)
SSH_AUTH_SUCCESS Accepted %{WORD:ssh_authmethod} for %{USERNAME:ssh_user} from
%{IP:ssh_client_ip} port %{NUMBER:ssh_client_port} (?<ssh_protocol>\w+\d+)(?:%{WORD:ssh_pubkey_type} %{GREEDYDATA:ssh_pubkey_fingerprint})?
SSH_DISCONNECT Received disconnect from %{IP:ssh_client_ip} port
%{INT:ssh_client_port}.*?:\s+ %{GREEDYDATA:ssh_disconnect_reason}
```

86.13. JSON (xm_json)

This module provides functions and procedures for processing data formatted as **JSON**. JSON can be generated from log data, or JSON can be parsed into **fields**. Unfortunately, the JSON specification does not define a type for datetime values so these are represented as JSON strings. The JSON parser in `xm_json` can automatically detect

datetime values, so it is not necessary to explicitly use [parsedate\(\)](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.13.1. Configuration

The *xm_json* module accepts the following directives in addition to the [common module directives](#).

DateFormat

This optional directive does NOT override the global [DateFormat](#). The format specified here only affects the datetime strings in the generated JSON.

If this directive is not specified, the default is `YYYY-MM-DD hh:mm:ss.sUTC`.

DetectNestedJSON

This optional directive can be used to disable the autodetection of nested JSON strings when calling [to_json\(\)](#). For example if a field named `key` contains the string value of `{"subkey":42}`, [to_json\(\)](#) would produce `{"key": {"subkey":42}}` which would be produced when this directive is set to `FALSE`. This autodetection causes it to produce `{"key": {"subkey":42}}` which is a valid nested JSON record. By default, [DetectNestedJSON](#) is `TRUE`.

Flatten

This optional boolean directive specifies that the [parse_json\(\)](#) procedure should flatten nested JSON, creating field names with dot notation. The default is `FALSE`. If [Flatten](#) is set to `TRUE`, the following JSON will populate the fields `$event.time` and `$event.severity`:

```
{"event": {"time": "2015-01-01T00:00:00.000Z", "severity": "ERROR"}}
```

ForceUTF8

This optional boolean directive specifies whether the generated JSON should be valid UTF-8. The JSON specification requires JSON records to be UTF-8 encoded, and some tools fail to parse JSON if it is not valid UTF-8. If [ForceUTF8](#) is set to `TRUE`, the generated JSON will be validated and any invalid character will be replaced with a question mark (?). The default is `FALSE`.

ParseDate

If this boolean directive is set to `TRUE`, *xm_json* will attempt to parse as a timestamp any string that appears to begin with a 4-digit year (as a regular expression, `^[\d]{4}[\d]{2}[\d]{2}-`). If this directive is set to `FALSE`, *xm_json* will not attempt to parse these strings. The default is `TRUE`.

PrettyPrint

If set to `TRUE`, this optional boolean directive specifies that the generated JSON should be pretty-printed, where each key-value is printed on a new indented line. Note that this adds line-breaks to the JSON records, which can cause parser errors in some tools that expect single-line JSON. If this directive is not specified, the default is `FALSE`.

UnFlatten

This optional boolean directive specifies that the [to_json\(\)](#) procedure should generate nested JSON when field names exist containing the dot (.). For example, if [UnFlatten](#) is set to `TRUE`, the two fields `$event.time` and `$event.severity` will be converted to JSON as follows:

```
{"event": {"time": "2015-01-01T00:00:00.000Z", "severity": "ERROR"}}
```

When [UnFlatten](#) is set to `FALSE` (the default if not specified), the following JSON would result:

```
{"event.time": "2015-01-01T00:00:00.000Z", "event.severity": "ERROR"}
```

86.13.2. Functions

The following functions are exported by xm_json.

`string to_json()`

Convert the fields to JSON and return this as a string value. The `$raw_event` field and any field having a leading dot (.) or underscore (_) will be automatically excluded.

86.13.3. Procedures

The following procedures are exported by xm_json.

`parse_json();`

Parse the `$raw_event` field as JSON input.

`parse_json(string source);`

Parse the given string as JSON format.

`to_json();`

Convert the fields to JSON and put this into the `$raw_event` field. The `$raw_event` field and any field having a leading dot (.) or underscore (_) will be automatically excluded.

86.13.4. Examples

Example 398. Syslog to JSON Format Conversion

The following configuration accepts Syslog (both BSD and IETF) via TCP and converts it to JSON.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension json>
6     Module xm_json
7 </Extension>
8
9 <Input tcp>
10    Module im_tcp
11    Port 1514
12    Host 0.0.0.0
13    Exec parse_syslog(); to_json();
14 </Input>
15
16 <Output file>
17     Module om_file
18     File "/var/log/json.txt"
19 </Output>
20
21 <Route tcp_to_file>
22     Path tcp => file
23 </Route>
```

Input Sample

```
<30>Sep 30 15:45:43 host44.localdomain.hu acpid: 1 client rule loaded
```

Output Sample

```
{
  "MessageSourceAddress": "127.0.0.1",
  "EventReceivedTime": "2011-03-08 14:22:41",
  "SyslogFacilityValue": 1,
  "SyslogFacility": "DAEMON",
  "SyslogSeverityValue": 5,
  "SyslogSeverity": "INFO",
  "SeverityValue": 2,
  "Severity": "INFO",
  "Hostname": "host44.localdomain.hu",
  "EventTime": "2011-09-30 14:45:43",
  "SourceName": "acpid",
  "Message": "1 client rule loaded "
```

Example 399. Converting Windows EventLog to Syslog-Encapsulated JSON

The following configuration reads the Windows EventLog and converts it to the BSD Syslog format, with the message part containing the fields in JSON.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension json>
6   Module      xm_json
7 </Extension>
8
9 <Input eventlog>
10  Module     im_msvisalog
11    Exec      $Message = to_json(); to_syslog_bsd();
12 </Input>
13
14 <Output tcp>
15   Module     om_tcp
16   Host       192.168.1.1
17   Port       1514
18 </Output>
19
20 <Route eventlog_json_tcp>
21   Path       eventlog => tcp
22 </Route>
```

Output Sample

```
<14>Mar  8 14:40:11 WIN-OUNNPISDHIG Service_Control_Manager: {"EventTime":"2012-03-08
14:40:11", "EventTimeWritten":"2012-03-08 14:40:11", "Hostname":"WIN-
OUNNPISDHIG", "EventType":"INFO", "SeverityValue":2, "Severity":"INFO", "SourceName":"Service
Control
Manager", "FileName":"System", "EventID":7036, "CategoryNumber":0, "RecordNumber":6788, "Message":"T
he nxlog service entered the running state. ", "EventReceivedTime":"2012-03-08 14:40:12"}<
```

86.14. Key-Value Pairs (xm_kvp)

This module provides functions and procedures for processing data formatted as key-value pairs (KVPs), also commonly called "name-value pairs". The module can both parse and generate key-value formatted data.

It is quite common to have a different set of keys in each log line when accepting key-value formatted input messages. Extracting values from such logs using regular expressions can be quite cumbersome. The *xm_kvp* extension module automates this process.

Log messages containing key-value pairs typically look like one the following:

- `key1: value1, key2: value2, key42: value42`
- `key1="value 1"; key2="value 2"`
- `Application=smtp, Event='Protocol Conversation', status='Client Request', ClientRequest='HELO 1.2.3.4'`

Keys are usually separated from the value using an equal sign (=) or a colon (:); and the key-value pairs are delimited with a comma (,), a semicolon (;), or a space. In addition, values and keys may be quoted and may contain escaping. The module will try to guess the format, or the format can be explicitly specified using the

configuration directives below.

NOTE

It is possible to use more than one *xm_kvp* module instance with different options in order to support different KVP formats at the same time. For this reason, functions and procedures exported by the module are public and must be referenced by the module instance name.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.14.1. Configuration

The *xm_kvp* module accepts the following directives in addition to the [common module directives](#).

EscapeChar

This optional directive takes a single character (see [below](#)) as argument. It specifies the character used for escaping special characters. The escape character is used to prefix the following characters: the **EscapeChar** itself, the **KeyQuoteChar**, and the **ValueQuoteChar**. If **EscapeControl** is TRUE, the newline (**\n**), carriage return (**\r**), tab (**\t**), and backspace (**\b**) control characters are also escaped. The default escape character is the backslash (****).

EscapeControl

If this optional boolean directive is set to TRUE, control characters are also escaped. See the **EscapeChar** directive for details. The default is TRUE (control characters are escaped). Note that this is necessary in order to support single-line KVP field lists containing line-breaks.

KeyQuoteChar

This optional directive takes a single character (see [below](#)) as argument. It specifies the quote character for enclosing key names. If this directive is not specified, the module will accept single-quoted keys, double-quoted keys, and unquoted keys.

KVDelimiter

This optional directive takes a single character (see [below](#)) as argument. It specifies the delimiter character used to separate the key from the value. If this directive is not set and the [parse_kvp\(\)](#) procedure is used, the module will try to guess the delimiter from the following: the colon (**:**) or the equal-sign (**=**).

KVPDelimiter

This optional directive takes a single character (see [below](#)) as argument. It specifies the delimiter character used to separate the key-value pairs. If this directive is not set and the [parse_kvp\(\)](#) procedure is used, the module will try to guess the delimiter from the following: the comma (**,**), the semicolon (**;**), or the space.

QuoteMethod

This directive can be used to specify the quote method used for the values by [to_kvp\(\)](#).

All

The values will be always quoted. This is the default.

Delimiter

The value will be only enclosed in quotes if it contains the delimiter character.

None

The values will not be quoted.

ValueQuoteChar

This optional directive takes a single character (see [below](#)) as argument. It specifies the quote character for enclosing key values. If this directive is not specified, the module will accept single-quoted values, double-quoted values, and unquoted values. Normally, quotation is used when the value contains a space or the

KVDelimiter character.

86.14.1.1. Specifying Quote, Escape, and Delimiter Characters

The **KeyQuoteChar**, **ValueQuoteChar**, **EscapeChar**, **KVDelimiter**, and **KVPDelimiter** directives can be specified in several ways.

Unquoted single character

Any printable character can be specified as an unquoted character, except for the backslash (\):

```
Delimiter ;
```

Control characters

The following non-printable characters can be specified with escape sequences:

\a

audible alert (bell)

\b

backspace

\t

horizontal tab

\n

newline

\v

vertical tab

\f

formfeed

\r

carriage return

For example, to use TAB delimiting:

```
Delimiter \t
```

A character in single quotes

The configuration parser strips whitespace, so it is not possible to define a space as the delimiter unless it is enclosed within quotes:

```
Delimiter ' '
```

Printable characters can also be enclosed:

```
Delimiter ';'
```

The backslash can be specified when enclosed within quotes:

```
Delimiter '\\'
```

A character in double quotes

Double quotes can be used like single quotes:

```
Delimiter " "
```

The backslash can be specified when enclosed within double quotes:

```
Delimiter "\\"
```

A hexadecimal ASCII code

Hexadecimal ASCII character codes can also be used by prepending `0x`. For example, the space can be specified as:

```
Delimiter 0x20
```

This is equivalent to:

```
Delimiter " "
```

86.14.2. Functions

The following functions are exported by `xm_kvp`.

`string to_kvp()`

Convert the internal fields to a single key-value pair formatted string.

86.14.3. Procedures

The following procedures are exported by `xm_kvp`.

`parse_kvp();`

Parse the `$raw_event` field as key-value pairs and populate the internal fields using the key names.

`parse_kvp(string source);`

Parse the given string key-value pairs and populate the internal fields using the key names.

`reset_kvp();`

Reset the KVP parser so that the autodetected `KeyQuoteChar`, `ValueQuoteChar`, `KVDelimiter`, and `KVPDelimiter` characters can be detected again.

`to_kvp();`

Format the internal fields as key-value pairs and put this into the `$raw_event` field.

86.14.4. Examples

The following examples illustrate various scenarios for parsing KVPs, whether embedded, encapsulated (in Syslog, for example), or alone. In each case, the logs are converted from KVP input files to JSON output files, though obviously there are many other possibilities.

Example 400. Simple KVP Parsing

The following two lines of input are in a simple KVP format where each line consists of various keys with values assigned to them.

Input Sample

```
Name=John, Age=42, Weight=84, Height=142
Name=Mike, Weight=64, Age=24, Pet=dog, Height=172
```

This input can be parsed with the following configuration. The parsed fields can be used in NXLog expressions: a new field named **\$Overweight** is added and set to TRUE if the conditions are met. Finally a few automatically added fields are removed, and the log is then converted to JSON.

nxlog.conf

```

1 <Extension kvp>
2   Module      xm_kvp
3   KVDelimiter ,
4   KVDelimiter =
5   EscapeChar  \\
6 </Extension>
7
8 <Extension json>
9   Module      xm_json
10 </Extension>
11
12 <Input filein>
13   Module      im_file
14   File        "modules/extension/kvp/xm_kvp5.in"
15   <Exec>
16     if $raw_event =~ /^#/ drop();
17     else
18     {
19       kvp->parse_kvp();
20       delete($EventReceivedTime);
21       delete($SourceModuleName);
22       delete($SourceModuleType);
23       if ( integer($Weight) > integer($Height) - 100 ) $Overweight = TRUE;
24       to_json();
25     }
26   </Exec>
27 </Input>
28
29 <Output fileout>
30   Module      om_file
31   File        'tmp/output'
32 </Output>
33
34 <Route parse_kvp>
35   Path        filein => fileout
36 </Route>
```

Output Sample

```
{"Name": "John", "Age": "42", "Weight": "84", "Height": "142", "Overweight": true}
{"Name": "Mike", "Weight": "64", "Age": "24", "Pet": "dog", "Height": "172"}
```

Example 401. Parsing KVPs in Cisco ACS Syslog

The following lines are from a Cisco ACS source.

Input Sample

```
<38>2010-10-12 21:01:29 10.0.1.1 CisACS_02_FailedAuth 1k1fg93nk 1 0 Message-Type=Authen
failed,User-Name=John,NAS-IP-Address=10.0.1.2,AAA Server=acs01<
<38>2010-10-12 21:01:31 10.0.1.1 CisACS_02_FailedAuth 2k1fg63nk 1 0 Message-Type=Authen
failed,User-Name=Foo,NAS-IP-Address=10.0.1.2,AAA Server=acs01<
```

These logs are in Syslog format with a set of values present in each record and an additional set of KVPs. The following configuration can be used to process this and convert it to JSON.

nxlog.conf

```
1 <Extension json>
2     Module      xm_json
3 </Extension>
4
5 <Extension syslog>
6     Module      xm_syslog
7 </Extension>
8
9 <Extension kvp>
10    Module      xm_kvp
11    KVDelimiter =
12    KVPDelimiter ,
13 </Extension>
14
15 <Input cisco>
16     Module      im_file
17     File        "modules/extension/kvp/cisco_acs.in"
18     <Exec>
19         parse_syslog_bsd();
20         if ( $Message =~ /^CisACS_(\d\d)_($+) ($+) (\d+) (\d+) (.*)$/ )
21         {
22             $ACSCategoryNumber = $1;
23             $ACSCategoryName = $2;
24             $ACSMessagId = $3;
25             $ACSTotalSegments = $4;
26             $ACSSegmentNumber = $5;
27             $Message = $6;
28             kvp->parse_kvp($Message);
29         }
30         else log_warning("does not match: " + to_json());
31     </Exec>
32 </Input>
33
34 <Output file>
35     Module      om_file
36     File        "tmp/output"
37     Exec        delete($EventReceivedTime);
38     Exec        to_json();
39 </Output>
40
41 <Route cisco_to_file>
42     Path        cisco => file
43 </Route>
```

Output Sample

```
{"SourceModuleName":"cisco", "SourceModuleType": "im_file", "SyslogFacilityValue":4, "SyslogFacility": "AUTH", "SyslogSeverityValue":6, "SyslogSeverity": "INFO", "SeverityValue":2, "Severity": "INFO", "Hostname": "10.0.1.1", "EventTime": "2010-10-12 21:01:29", "Message": "Message-Type=Authen failed,User-Name=John,NAS-IP-Address=10.0.1.2,AAA Server=acs01", "ACSCategoryNumber": "02", "ACSCategoryName": "FailedAuth", "ACSMessagId": "1k1fg93nk", "ACSTotalSegments": "1", "ACSSegmentNumber": "0", "Message-Type": "Authen failed", "User-Name": "John", "NAS-IP-Address": "10.0.1.2", "AAA Server": "acs01"}  
{"SourceModuleName":"cisco", "SourceModuleType": "im_file", "SyslogFacilityValue":4, "SyslogFacility": "AUTH", "SyslogSeverityValue":6, "SyslogSeverity": "INFO", "SeverityValue":2, "Severity": "INFO", "Hostname": "10.0.1.1", "EventTime": "2010-10-12 21:01:31", "Message": "Message-Type=Authen failed,User-Name=Foo,NAS-IP-Address=10.0.1.2,AAA Server=acs01", "ACSCategoryNumber": "02", "ACSCategoryName": "FailedAuth", "ACSMessagId": "2k1fg63nk", "ACSTotalSegments": "1", "ACSSegmentNumber": "0", "Message-Type": "Authen failed", "User-Name": "Foo", "NAS-IP-Address": "10.0.1.2", "AAA Server": "acs01"}
```

Example 402. Parsing KVPs in Sidewinder Logs

The following line is from a Sidewinder log source.

Input Sample

```
date="May 5 14:34:40 2009
MDT",fac=f_mail_filter,area=a_kmvfilter,type=t_mimevirus_reject,pri=p_major,pid=10174,ruid=0,eu
id=0,pgid=10174,logid=0,cmd=kmvfilter,domain=MMF1,edomain=MMF1,message_id=(null),srcip=66.74.18
4.9,mail_sender=<habuzeid6@...>,virus_name=W32/Netsky.c@MM!zip,reason="Message scan detected a
Virus in msg Unknown, message being Discarded, and not quarantined"↵
```

This can be parsed and converted to JSON with the following configuration.

nxlog.conf

```
1 <Extension kvp>
2   Module      xm_kvp
3   KVPDelimiter ,
4   KVDelimiter =
5   EscapeChar  \\
6   ValueQuoteChar "
7 </Extension>
8
9 <Extension json>
10  Module      xm_json
11 </Extension>
12
13 <Input sidewinder>
14   Module      im_file
15   File        "modules/extension/kvp/sidewinder.in"
16   Exec        kvp->parse_kvp(); delete($EventReceivedTime); to_json();
17 </Input>
18
19 <Output file>
20   Module      om_file
21   File        'tmp/output'
22 </Output>
23
24 <Route sidewinder_to_file>
25   Path        sidewinder => file
26 </Route>
```

Output Sample

```
{"SourceModuleName":"sidewinder","SourceModuleType":"im_file","date":"May 5 14:34:40 2009 MDT"
,"fac":"f_mail_filter","area":"a_kmvfilter","type":"t_mimevirus_reject","pri":"p_major","pid":"
10174","ruid":"0","euid":"0","pgid":10174,"logid":0,"cmd": "kmvfilter","domain": "MMF1","edom
ain": "MMF1","message_id": "(null)","srcip": "66.74.184.9","mail_sender": "<habuzeid6@...>","virus_na
me": "W32/Netsky.c@MM!zip","reason": "Message scan detected a Virus in msg Unknown, message being
Discarded, and not quarantined"}
```

Example 403. Parsing URL Request Parameters in Apache Access Logs

URLs in HTTP requests frequently contain URL parameters which are a special kind of key-value pairs delimited by the ampersand (`&`). Here is an example of two HTTP requests logged by the Apache web server in the Combined Log Format.

Input Sample

```
192.168.1.1 - foo [11/Jun/2013:15:44:34 +0200] "GET /do?action=view&obj_id=2 HTTP/1.1" 200 1514
"https://localhost" "Mozilla/5.0 (X11; Linux x86_64; rv:17.0) Gecko/17.0 Firefox/17.0"-
192.168.1.1 - - [11/Jun/2013:15:44:44 +0200] "GET /do?action=delete&obj_id=42 HTTP/1.1" 401 788
"https://localhost" "Mozilla/5.0 (X11; Linux x86_64; rv:17.0) Gecko/17.0 Firefox/17.0"-

```

The following configuration file parses the access log and extracts all the fields. The request parameters are extracted into the \$HTTPParams field using a regular expression, and then this field is further parsed using the KVP parser. At the end of the processing all fields are converted to KVP format using the `to_kvp()` procedure of the `kvp2` instance.

nxlog.conf

```
1 <Extension kvp>
2   Module      xm_kvp
3   KVPDelimiter &
4   KVDelimiter   =
5 </Extension>
6
7 <Extension kvp2>
8   Module      xm_kvp
9   KVPDelimiter ;
10  KVDelimiter   =
11  #QuoteMethod None
12 </Extension>
13
14 <Input apache>
15  Module      im_file
16  File        "modules/extension/kvp/apache_url.in"
17  <Exec>
18    if $raw_event =~ /(?x)^(\S+)\ (\S+)\ (\S+)\ \[(\[\^\]]+)\]\ \"(\S+)\ (.+)
19      \ HTTP.\d\.\d\" \ (\d+)\ (\d+)\ \"([^\"]+)\\" \ \"([^\"]+)\\" /
20    {
21      $Hostname = $1;
22      if $3 != '-' $AccountName = $3;
23      $EventTime = parsedate($4);
24      $HTTPMethod = $5;
25      $HTTPURL = $6;
26      $HTTPResponseStatus = $7;
27      $FileSize = $8;
28      $HTTPReferer = $9;
29      $HTTPUserAgent = $10;
30      if $HTTPURL =~ /\?(.+)/ { $HTTPParams = $1; }
31      kvp->parse_kvp($HTTPParams);
32      delete($EventReceivedTime);
33      kvp2->to_kvp();
34    }
35  </Exec>
36 </Input>
37
38 <Output file>
39  Module      om_file
40  File        'tmp/output'
41 </Output>
42
43 <Route apache_to_file>
44  Path        apache => file
45 </Route>
```

The two request parameters `action` and `obj_id` then appear at the end of the KVP formatted lines.

Output Sample

```
SourceModuleName=apache;SourceModuleType=im_file;Hostname=192.168.1.1;AccountName=foo;EventTime=2013-06-11  
15:44:34;HTTPMethod=GET;HTTPURL=/do?action=view&obj_id=2;HTTPResponseStatus=200;FileSize=1514;HTTPReferer=https://localhost;HTTPUserAgent='Mozilla/5.0 (X11; Linux x86_64; rv:17.0) Gecko/17.0 Firefox/17.0';HTTPParams=action=view&obj_id=2;action=view;obj_id=2;  
SourceModuleName=apache;SourceModuleType=im_file;Hostname=192.168.1.1;EventTime=2013-06-11  
15:44:44;HTTPMethod=GET;HTTPURL=/do?action=delete&obj_id=42;HTTPResponseStatus=401;FileSize=788;HTTPReferer=https://localhost;HTTPUserAgent='Mozilla/5.0 (X11; Linux x86_64; rv:17.0) Gecko/17.0 Firefox/17.0';HTTPParams=action=delete&obj_id=42;action=delete;obj_id=42;
```

NOTE

URL escaping is not handled.

86.15. LEEF (xm_leef)

This module provides two functions to generate and parse data in the Log Event Extended Format (LEEF), which is used by IBM Security QRadar products. For more information about the format see the [Log Event Extended Format \(LEEF\) Version 2](#) specification.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.15.1. Configuration

The *xm_leef* module accepts the following directives in addition to the [common module directives](#).

AddSyslogHeader

This optional boolean directive specifies whether a RFC 3164 (BSD-style) Syslog header should be prepended to the output. This defaults to TRUE (a Syslog header will be added by the [to_leef\(\)](#) procedure).

LEEFHeader

This optional directive takes a [string](#) type [expression](#) and only has an effect on how [to_leef\(\)](#) formats the result. It should evaluate to the following format:

```
|LEEF:1.0|Microsoft|MSEExchange|2013 SP1|15345|
```

It should typically be used as follows:

```
LEEFHeader 'LEEF:1.0|Microsoft|MSEExchange|2013 SP1|' + $EventID + '|'
```

When this directive is not specified, the LEEF header is constructed using the [\\$Vendor](#), [\\$SourceName](#) (or [\\$SourceModuleName](#)), [\\$Version](#), and [\\$EventID](#) fields.

86.15.2. Functions

The following functions are exported by xm_leef.

[*string to_leef\(\)*](#)

Convert the internal fields to a single LEEF formatted string.

86.15.3. Procedures

The following procedures are exported by xm_leef.

`parse_leef();`

Parse the `$raw_event` field as key-value pairs and populate the internal fields using the key names.

`parse_leef(string source);`

Parse the given string key-value pairs and populate the internal fields using the key names.

`to_leef();`

Format the internal fields as LEEF and put this into the `$raw_event` field.

86.15.4. Examples

Example 404. Sending Windows EventLog as LEEF over UDP

This configuration will collect Windows EventLog and NXLog internal messages, convert them to LEEF, and forward via UDP.

nxlog.conf

```

1 <Extension leef>
2   Module xm_leef
3 </Extension>
4
5 <Input internal>
6   Module im_internal
7 </Input>
8
9 <Input eventlog>
10  Module im_msvisatalog
11 </Input>
12
13 <Output udp>
14   Module om_udp
15   Host   192.168.168.2
16   Port   1514
17   Exec   to_leef();
18 </Output>
19
20 <Route qradar>
21   Path   internal, eventlog => udp
22 </Route>
```

86.16. Microsoft DNS Server (xm_msdns)

This module provides support for parsing Windows DNS Server logs. An `InputType` is registered using the name of the extension module instance. For special cases, the `parse_msdns()` procedure can be used instead for parsing individual events or strings.

WARNING

The `xm_msdns` module does not support the detailed format enabled via the **Details** option in the DNS Server **Debug Logging** configuration. NXLog could be configured to parse this format with the `xm_multiline` module.

86.16.1. Configuration

The `xm_msdns` module accepts the following directives in addition to the [common module directives](#).

EventLine

This boolean directive specifies **EVENT** lines in the input should be parsed. If set to FALSE, **EVENT** lines will be discarded. The default is TRUE.

NoteLine

This boolean directive specifies that **Note:** lines in the input should be parsed. If set to FALSE, **Note:** lines will be discarded. The default is TRUE.

PacketLine

This boolean directive specifies that **PACKET** lines in the input should be parsed. If set to FALSE, **PACKET** lines will be discarded. The default is TRUE.

86.16.2. Procedures

The following procedures are exported by xm_msdns.

parse_msdns();

Parse the **\$raw_event** field and populate the DNS log fields.

parse_msdns(string source);

Parse the given string and populate the DNS log fields.

86.16.3. Fields

The following fields are used by xm_msdns.

\$raw_event (type: string)

The raw string from the event.

\$AuthoritativeAnswer (type: boolean)

For PACKET events, set to TRUE if the "Authoritative Answer" flag is set.

\$Context (type: string)

The event type, one of **PACKET**, **EVENT**, or **Note**.

\$EventDescription (type: string)

The description for EVENT type events.

\$EventTime (type: datetime)

The timestamp of the event.

\$FlagsHex (type: string)

The flags in hexadecimal, for PACKET events only.

\$InternalPacketIdentifier (type: string)

For PACKET events, an internal ID corresponding with the event.

\$Note (type: string)

For "Note" type events, this field contains the note.

\$Opcodes (type: string)

One of **Standard**, **Query**, **Notify**, **Update**, and **Unknown**; for PACKET events.

\$Protocol (*type: string*)

The protocol being used; one of **TCP** or **UDP**. This field is added for the PACKET type only.

\$QueryResponseIndicator (*type: string*)

This field indicates whether a PACKET event corresponds with a query or a response, and is set to either **Query** or **Response**.

\$QuestionName (*type: string*)

The lookup value for PACKET; for example **example.com**.

\$QuestionType (*type: string*)

The lookup type for PACKET events; for example, **A** or **AAAA**.

\$RecursionAvailable (*type: boolean*)

For PACKET events, set to TRUE if the "Recursion Available" flag is set.

\$RecursionDesired (*type: boolean*)

For PACKET events, set to TRUE if the "Recursion Desired" flag is set.

\$RemoteIP (*type: string*)

The IP address of the requesting client, for PACKET events only.

\$ResponseCode (*type: string*)

For PACKET events, the DNS Server response code.

\$SendReceiveIndicator (*type: string*)

This field indicates the direction for a PACKET event, and is set to either **Snd** or **Rcv**.

\$ThreadId (*type: string*)

The ID of the thread that produced the event.

\$TruncatedResponse (*type: boolean*)

For PACKET events, set to TRUE if the "Truncated Response" flag is set.

\$Xid (*type: string*)

For PACKET events, the hexadecimal XID.

86.16.4. Examples

Example 405. Parsing DNS Logs With InputType

In this configuration, the DNS log file at `C:\dns.log` is parsed using the `InputType` provided by the `xm_msdns` module. Any **Note:** lines in the input are discarded (the `NoteLine` directive is set to FALSE).

`nxlog.conf`

```

1 <Extension dns_parser>
2   Module      xm_msdns
3   EventLine   TRUE
4   PacketLine  TRUE
5   NoteLine    FALSE
6 </Extension>
7
8 <Input in>
9   Module      im_file
10  File        'modules/extension/msdns/xm_msdns1.in'
11  InputType   dns_parser
12 </Input>
```

Example 406. Parsing DNS Logs With parse_msdns()

For cases where parsing via `InputType` is not possible, individual events can be parsed with the `parse_msdns()` procedure.

`nxlog.conf`

```

1 <Extension dns_parser>
2   Module      xm_msdns
3 </Extension>
4
5 <Input in>
6   Module      im_file
7   File        'modules/extension/msdns/xm_msdns1.out'
8   Exec        dns_parser->parse_msdns();
9 </Input>
```

86.17. Multi-Line Message Parser (xm_multiline)

This module can be used for parsing log messages that span multiple lines. All lines in an event are joined to form a single NXLog event record, which can be further processed as required. Each multi-line event is detected through some combination of header lines, footer lines, and fixed line counts, as configured. The name of the `xm_multiline` module instance is specified by the input module's `InputType` directive.

The module maintains a separate context for each input source, allowing multi-line messages to be processed correctly even when coming from multiple sources (specifically, multiple files or multiple network connections).

WARNING

UDP is treated as a single source and all logs are processed under the same context. It is therefore not recommended to use this module with `im_udp` if messages will be received by multiple UDP senders (such as Syslog).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.17.1. Configuration

The `xm_multiline` module accepts the following directives in addition to the [common module directives](#). One of

[FixedLineCount](#) and [HeaderLine](#) must be specified.

FixedLineCount

This directive takes a positive integer number defining the number of lines to concatenate. This is useful when receiving log messages spanning a fixed number of lines. When this number is defined, the module knows where the event message ends and will not hold a message in the buffers until the next message arrives.

HeaderLine

This directive takes a [string](#) or a [regular expression](#) literal. This will be matched against each line. When the match is successful, the successive lines are appended until the next header line is read. This directive is mandatory unless [FixedLineCount](#) is used.

NOTE

Until a new message arrives with its associated header, the previous message is stored in the buffers because the module does not know where the message ends. The [im_file](#) module will forcibly flush this buffer after the configured [PollInterval](#) timeout. If this behavior is unacceptable, disable [AutoFlush](#), use an end marker with [EndLine](#), or switch to an encapsulation method (such as JSON).

AutoFlush

If set to TRUE, this boolean directive specifies that the corresponding *im_file* module should forcibly flush the buffer after its configured [PollInterval](#) timeout. The default is TRUE. If [EndLine](#) is used, [AutoFlush](#) is automatically set to FALSE to disable this behavior.

EndLine

This is similar to the [HeaderLine](#) directive. This optional directive also takes a [string](#) or a [regular expression](#) literal to be matched against each line. When the match is successful the message is considered complete.

Exec

This directive is almost identical to the behavior of the [Exec](#) directive used by the other modules with the following differences:

- each line is passed in [\\$raw_event](#) as it is read, and the line terminator is included; and
- other fields cannot be used, and captured strings can not be stored as separate fields.

This is mostly useful for rewriting lines or filtering out certain lines with the [drop\(\)](#) procedure.

86.17.2. Examples

Example 407. Parsing multi-line XML logs and converting to JSON

XML is commonly formatted as indented multi-line to make it more readable. In the following configuration file the [HeaderLine](#) and [EndLine](#) directives are used to parse the events. The events are then converted to JSON after some timestamp normalization.

nxlog.conf

```
1 <Extension multiline>
2   Module      xm_multiline
3   HeaderLine  /^<event>/
4   EndLine     /^</event>/
5 </Extension>
6
7 <Extension xmlparser>
8   Module      xm_xml
9 </Extension>
10
11 <Extension json>
12   Module      xm_json
13 </Extension>
14
15 <Input filein>
16   Module      im_file
17   File        "modules/extension/multiline/xm_multiline5.in"
18   InputType   multiline
19   <Exec>
20     # Discard everything that doesn't seem to be an xml event
21     if $raw_event !~ /^<event>/ drop();
22
23     # Parse the xml event
24     parse_xml();
25
26     # Rewrite some fields
27     $EventTime = parsedate($timestamp);
28     delete($timestamp);
29     delete($EventReceivedTime);
30
31     # Convert to JSON
32     to_json();
33   </Exec>
34 </Input>
35
36 <Output fileout>
37   Module      om_file
38   File        'tmp/output'
39 </Output>
40
41 <Route parse_xml>
42   Path       filein => fileout
43 </Route>
```

Input Sample

```
<?xml version="1.0" encoding="UTF-8">
<event>
  <timestamp>2012-11-23 23:00:00</timestamp>
  <severity>ERROR</severity>
  <message>
    Something bad happened.
    Please check the system.
  </message>
</event>
<event>
  <timestamp>2012-11-23 23:00:12</timestamp>
  <severity>INFO</severity>
  <message>
    System state is now back to normal.
  </message>
</event>
```

Output Sample

```
{"SourceModuleName":"filein", "SourceModuleType":"im_file", "severity":"ERROR", "message": "\nSomething bad happened.\n    Please check the system.\n  ", "EventTime": "2012-11-23 23:00:00"}\n{"SourceModuleName":"filein", "SourceModuleType":"im_file", "severity":"INFO", "message": "\nSystem state is now back to normal.\n  ", "EventTime": "2012-11-23 23:00:12"}
```

Example 408. Parsing DICOM Logs

Each log message has a header (TIMESTAMP INTEGER SEVERITY) which is used as the message boundary. A regular expression is defined for this with the [HeaderLine](#) directive. Each log message is prepended with an additional line containing dashes and is written to a file.

nxlog.conf

```
1 <Extension dicom_multi>
2   Module           xm_multiline
3   HeaderLine      /^\d\d\d\d-\d\d\d\d:\d\d\d\d:\d\d.\d+\s+\d+\s+\S+\s+/
4 </Extension>
5
6 <Input filein>
7   Module          im_file
8   File            "modules/extension/multiline/xm_multiline4.in"
9   InputType       dicom_multi
10 </Input>
11
12 <Output fileout>
13   Module          om_file
14   File            'tmp/output'
15   Exec            $raw_event = "-----\n" + $raw_event;
16 </Output>
17
18 <Route parse_dicom>
19   Path            filein => fileout
20 </Route>
```

Input Sample

```
2011-12-1512:22:51.000000 4296 INFO Association Request Parameters:  
Our Implementation Class UID: 2.16.124.113543.6021.2  
Our Implementation Version Name: RZDCX_2_0_1_8  
Their Implementation Class UID:  
Their Implementation Version Name:  
Application Context Name: 1.2.840.10008.3.1.1.1  
Requested Extended Negotiation: none  
Accepted Extended Negotiation: none  
2011-12-1512:22:51.000000 4296 DEBUG Constructing Associate RQ PDU  
2011-12-1512:22:51.000000 4296 DEBUG WriteToConnection, length: 310, bytes written: 310,  
loop no: 1  
2011-12-1512:22:51.015000 4296 DEBUG PDU Type: Associate Accept, PDU Length: 216 + 6 bytes  
PDU header:  
02 00 00 00 d8 00 01 00 00 50 41 43 53 20 20  
20 20 20 20 20 20 20 20 20 52 5a 44 43 58 20  
20 20 20 20 20 20 20 20 20 00 00 00 00 00 00  
2011-12-1512:22:51.031000 4296 DEBUG DIMSE sendDcmDataset: sending 146 bytes
```

Output Sample

```
-----  
2011-12-1512:22:51.000000 4296 INFO Association Request Parameters:  
Our Implementation Class UID: 2.16.124.113543.6021.2  
Our Implementation Version Name: RZDCX_2_0_1_8  
Their Implementation Class UID:  
Their Implementation Version Name:  
Application Context Name: 1.2.840.10008.3.1.1.1  
Requested Extended Negotiation: none  
Accepted Extended Negotiation: none  
-----  
2011-12-1512:22:51.000000 4296 DEBUG Constructing Associate RQ PDU  
-----  
2011-12-1512:22:51.000000 4296 DEBUG WriteToConnection, length: 310, bytes written: 310,  
loop no: 1  
-----  
2011-12-1512:22:51.015000 4296 DEBUG PDU Type: Associate Accept, PDU Length: 216 + 6 bytes  
PDU header:  
02 00 00 00 d8 00 01 00 00 50 41 43 53 20 20  
20 20 20 20 20 20 20 20 20 52 5a 44 43 58 20  
20 20 20 20 20 20 20 20 20 00 00 00 00 00 00  
-----  
2011-12-1512:22:51.031000 4296 DEBUG DIMSE sendDcmDataset: sending 146 bytes
```

Example 409. Multi-line messages with a fixed string header

The following configuration will process messages having a fixed string header containing dashes. Each event is then prepended with a hash mark (#) and written to a file.

nxlog.conf

```

1 <Extension multiline>
2   Module      xm_multiline
3   HeaderLine  "-----"
4 </Extension>
5
6 <Input filein>
7   Module      im_file
8   File        "modules/extension/multiline/xm_multiline1.in"
9   InputType   multiline
10  Exec        $raw_event = "#" + $raw_event;
11 </Input>
12
13 <Output fileout>
14  Module      om_file
15  File        'tmp/output'
16 </Output>
17
18 <Route parse_multiline>
19  Path        filein => fileout
20 </Route>
```

Input Sample

```

-----+
1<
-----+
1<
2<
-----+
aaaaaaaaaaaaaaaaaaaaaaaaaa<
bbbbbbbbbbbbbbbbbbbbbbbbbb<
cccccccccccccccccccccccccc<
dddd<
-----+
```

Output Sample

```

#-----+
1<
#-----+
1<
2<
#-----+
aaaaaaaaaaaaaaaaaaaaaaaaa<
bbbbbbbbbbbbbbbbbbbbbbbbbb<
cccccccccccccccccccccccccc<
dddd<
#-----+
```

Example 410. Multi-line messages with fixed line count

The following configuration will process messages having a fixed line count of four. Lines containing only whitespace are ignored and removed. Each event is then prepended with a hash mark (#) and written to a file.

nxlog.conf

```

1 <Extension multiline>
2   Module      xm_multiline
3   FixedLineCount 4
4   Exec        if $raw_event =~ /^ \s*$/ drop();
5 </Extension>
6
7 <Input filein>
8   Module      im_file
9   File        "modules/extension/multiline/xm_multiline2.in"
10  InputType   multiline
11 </Input>
12
13 <Output fileout>
14   Module      om_file
15   File        'tmp/output'
16   Exec        $raw_event = "#" + $raw_event;
17 </Output>
18
19 <Route parse_multiline>
20   Path        filein => fileout
21 </Route>
```

Input Sample

```

1↵
2↵
3↵
4↵
1asd↵
↵
2asdassad↵
3ewrwerew↵
4xcbccvbc↵
↵
1dsfsdfsdfsd↵
2sfdsfsdrewrwe↵
↵
3sdfsdfsew↵
4werwerwrwe↵
```

Output Sample

```

#1↵
2↵
3↵
4↵
#1asd↵
2asdassad↵
3ewrwerew↵
4xcbccvbc↵
#1dsfsdfsdfsd↵
2sfdsfsdrewrwe↵
3sdfsdfsew↵
4werwerwrwe↵
```

Example 411. Multi-line messages with a Syslog header

Often, multi-line messages are logged over Syslog and each line is processed as an event, with its own Syslog header. It is commonly necessary to merge these back into a single event message.

Input Sample

Nov 21 11:40:27 hostname app[26459]: Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-
ERR TX-DRP TX-OVR Flg←								
Nov 21 11:40:27 hostname app[26459]: eth2	1500	0	16936814		0	0	0	30486067
0 8 0 BMRU←								
Nov 21 11:40:27 hostname app[26459]: lo	16436	0	277217234		0	0	0	
277217234 0 0 0 LRU←								
Nov 21 11:40:27 hostname app[26459]: tun0	1500	0	316943		0	0	0	368642
0 0 0 MOPRU←								
Nov 21 11:40:28 hostname app[26459]: Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-
ERR TX-DRP TX-OVR Flg←								
Nov 21 11:40:28 hostname app[26459]: eth2	1500	0	16945117		0	0	0	30493583
0 8 0 BMRU←								
Nov 21 11:40:28 hostname app[26459]: lo	16436	0	277217234		0	0	0	
277217234 0 0 0 LRU←								
Nov 21 11:40:28 hostname app[26459]: tun0	1500	0	316943		0	0	0	368642
0 0 0 MOPRU←								
Nov 21 11:40:29 hostname app[26459]: Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-
ERR TX-DRP TX-OVR Flg←								
Nov 21 11:40:29 hostname app[26459]: eth2	1500	0	16945270		0	0	0	30493735
0 8 0 BMRU←								
Nov 21 11:40:29 hostname app[26459]: lo	16436	0	277217234		0	0	0	
277217234 0 0 0 LRU←								
Nov 21 11:40:29 hostname app[26459]: tun0	1500	0	316943		0	0	0	368642
0 0 0 MOPRU←								

The following configuration strips the Syslog header from the netstat output stored in the traditional Syslog formatted file, and each message is then printed again with a line of dashes used as a separator.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension netstat>
6   Module      xm_multiline
7   FixedLineCount 4
8   <Exec>
9     parse_syslog_bsd();
10    $raw_event = $Message + "\n";
11  </Exec>
12 </Extension>
13
14 <Input filein>
15   Module      im_file
16   File        "modules/extension/multiline/xm_multiline3.in"
17   InputType   netstat
18 </Input>
19
20 <Output fileout>
21   Module      om_file
22   File        'tmp/output'
23   <Exec>
24     $raw_event = "-----" +
25     "-----\n" + $raw_event;
26   </Exec>
27 </Output>
28
29 <Route parse_multiline>
30   Path        filein => fileout
31 </Route>

```

Output Sample

Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth2	1500	0	16936814	0	0	0	30486067	0	8	0	BMRU
lo	16436	0	277217234	0	0	0	277217234	0	0	0	LRU
tun0	1500	0	316943	0	0	0	368642	0	0	0	MOPRU
<hr/>											
Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth2	1500	0	16945117	0	0	0	30493583	0	8	0	BMRU
lo	16436	0	277217234	0	0	0	277217234	0	0	0	LRU
tun0	1500	0	316943	0	0	0	368642	0	0	0	MOPRU
<hr/>											
Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth2	1500	0	16945270	0	0	0	30493735	0	8	0	BMRU
lo	16436	0	277217234	0	0	0	277217234	0	0	0	LRU
tun0	1500	0	316943	0	0	0	368642	0	0	0	MOPRU

86.18. NetFlow (xm_netflow)

This module provides a parser for NetFlow payload collected over UDP using [im_udp](#). It supports the following NetFlow protocol versions: v1, v5, v7, v9, and IPFIX.

NOTE

This module only supports parsing NetFlow data received as UDP datagrams and does not support TCP.

The module exports an input parser which can be referenced in the UDP input instance with the [InputType](#) directive:

InputType netflow

This input reader function parses the payload and extracts NetFlow specific fields.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.18.1. Configuration

The *xm_netflow* module accepts only the [common module directives](#).

86.18.2. Fields

The fields generated by *xm_netflow* are provided separately. Please refer to the documentation available online or in the NXLog package.

86.18.3. Examples

Example 412. Parsing UDP NetFlow Data

The following configuration receives NetFlow data over UDP and converts the parsed data into JSON.

nxlog.conf

```
1 <Extension netflow>
2   Module      xm_netflow
3 </Extension>
4
5 <Extension json>
6   Module      xm_json
7 </Extension>
8
9 <Input udpin>
10  Module     im_udp
11  Host       0.0.0.0
12  Port       2162
13  InputType  netflow
14 </Input>
15
16 <Output out>
17  Module     om_file
18  File       "netflow.log"
19  Exec       to_json();
20 </Output>
21
22 <Route nf>
23  Path       udpin => out
24 </Route>
```

86.19. NPS (xm_nps)

This module provides functions and procedures for processing data in NPS Database Format stored in files by Microsoft Radius services. Internet Authentication Service (IAS) is the Microsoft implementation of a RADIUS server and proxy. Internet Authentication Service (IAS) was renamed to Network Policy Server (NPS) starting with Windows Server 2008. This module is capable of parsing both IAS and NPS formatted data.

NPS formatted data typically looks like the following:

```
"RasBox", "RAS", 10/22/2006,09:13:09,1,"DOMAIN\user","DOMAIN\user",,,,,"192.168.132.45",12,,,"192.168.  
132.45",,,,0,"CONNECT 24000",1,2,4,,0,"311 1 192.168.132.45 07/31/2006 21:35:14  
749",.....,"MSRASV5.00",311,,  
"RasBox", "RAS", 10/22/2006,09:13:09,3,, "DOMAIN\user",.....,4,,36,"311 1 192.168.132.45  
07/31/2006 21:35:14 749",.....,"0x00453D36393120523D3020563D33",,,  
"RasBox", "RAS", 10/22/2006,09:13:13,1,"DOMAIN\user","DOMAIN\user",,,,,"192.168.132.45",12,,,"192.168.  
132.45",,,,0,"CONNECT 24000",1,2,4,,0,"311 1 192.168.132.45 07/31/2006 21:35:14  
750",.....,"MSRASV5.00",311,,,
```

For more information on the NPS format see the [Interpret NPS Database Format Log Files](#) article on Microsoft TechNet.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.19.1. Configuration

The `xm_nps` module accepts only the [common module directives](#).

86.19.2. Procedures

The following procedures are exported by `xm_nps`.

`parse_nps();`

Parse the `$raw_event` field as NPS input.

`parse_nps(string source);`

Parse the given string as NPS format.

86.19.3. Examples

Example 413. Parsing NPS Data

The following configuration reads NPS formatted files and converts the parsed data into JSON.

nxlog.conf

```

1 <Extension nps>
2   Module      xm_nps
3 </Extension>
4
5 <Extension json>
6   Module      xm_json
7 </Extension>
8
9 <Input filein>
10  Module     im_file
11  File       'C:\logs\IAS.log'
12  Exec       parse_nps();
13 </Input>
14
15 <Output fileout>
16  Module     om_file
17  File       'C:\out.json'
18  Exec       to_json();
19 </Output>
20
21 <Route nps_to_json>
22  Path       filein => fileout
23 </Route>
```

86.20. Pattern Matcher (*xm_pattern*)

This module makes it possible to execute pattern matching with a pattern database file in XML format. Using *xm_pattern* is more efficient than having NXLog regular expression rules listed in *Exec* directives, because it was designed in such a way that patterns do not need to be matched linearly. Regular expression sub-capturing can be used to set additional fields in the event record and arbitrary fields can be added under the scope of a pattern match for message classification. In addition, the module does an automatic on-the-fly pattern reordering internally for further speed improvements.

There are other techniques such as the radix tree which solve the linearity problem; the drawback is that usually these require the user to learn a special syntax for specifying patterns. If the log message is already parsed and is not treated as single line of message, then it is possible to process only a subset of the patterns which partially solves the linearity problem. With other performance improvements employed within the *xm_pattern* module, its speed can compare to the other techniques. Yet the *xm_pattern* module uses regular expressions which are familiar to users and can easily be migrated from other tools.

Traditionally, pattern matching on log messages has employed a technique where the log message was one string and the pattern (regular expression or radix tree based pattern) was executed against it. To match patterns against logs which contain structured data (such as the Windows EventLog), this structured data (the fields of the log) must be converted to a single string. This is a simple but inefficient method used by many tools.

The NXLog patterns defined in the XML pattern database file can contain more than one field. This allows multi-dimensional pattern matching. Thus with NXLog's *xm_pattern* module there is no need to convert all fields into a single string as it can work with multiple fields.

Patterns can be grouped together under pattern groups. Pattern groups serve an optimization purpose. The group can have an optional *matchfield* block which can check a condition. If the condition (such as **\$SourceName** matches **sshd**) is satisfied, the *xm_pattern* module will descend into the group and check each pattern against the

log. If the pattern group's condition did not match (`$SourceName` was not `sshd`), the module can skip all patterns in the group without having to check each pattern individually.

When the `xm_pattern` module finds a matching pattern, the `$PatternID` and `$PatternName` fields are set on the log message. These can be used later in conditional processing and correlation rules of the `pm_evcorr` module, for example.

NOTE

The `xm_pattern` module does not process all patterns. It exits after the first matching pattern is found. This means that at most one pattern can match a log message. Multiple patterns that can match the same subset of logs should be avoided. For example, with two regular expression patterns `^\d+` and `^\d\d`, only one will be matched but not consistently because the internal order of patterns and pattern groups is changed dynamically by `xm_pattern` (patterns with the highest match count are placed and tried first). For a strictly linearly executing pattern matcher, see the `Exec` directive.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.20.1. Configuration

The `xm_pattern` module accepts the following directives in addition to the [common module directives](#).

PatternFile

This mandatory directive specifies the name of the [pattern database file](#).

86.20.2. Functions

The following functions are exported by `xm_pattern`.

`boolean match_pattern()`

Execute the `match_pattern()` procedure. If the event is successfully matched, return TRUE, otherwise FALSE.

86.20.3. Procedures

The following procedures are exported by `xm_pattern`.

`match_pattern();`

Attempt to match the current event according to the [PatternFile](#). Execute statements and add fields as specified.

86.20.4. Fields

The following fields are used by `xm_pattern`.

`$PatternID` (type: `integer`)

The ID of the pattern that matched the event.

`$PatternName` (type: `string`)

The name of the pattern that matched the event.

86.20.5. Examples

Example 414. Using the match_pattern() Procedure

This configuration reads Syslog messages from file and parses them with `parse_syslog()`. The events are

then further processed with a pattern file and the corresponding `match_pattern()` procedure to add additional fields to SSH authentication success or failure events. The matching is done against the `$SourceName` and `$Message` fields, so the Syslog parsing must be performed before the pattern matching will work.

`nxlog.conf`

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension pattern>
6   Module      xm_pattern
7   PatternFile modules/extension/pattern/patterndb2,3.xml
8 </Extension>
9
10 <Input in>
11   Module     im_file
12   File       'test2.log'
13   <Exec>
14     parse_syslog();
15     match_pattern();
16   </Exec>
17 </Input>
```

The following pattern database contains two patterns to match SSH authentication messages. The patterns are under a group named `ssh` which checks whether the `$SourceName` field is `sshd` and only tries to match the patterns if the logs are indeed from `sshd`. The patterns both extract `$AuthMethod`, `$AccountName`, and `$SourceIPAddress` fields from the log message when the pattern matches the log. Additionally `$TaxonomyStatus` and `$TaxonomyAction` are set. The second pattern shows an `Exec` block example, which is evaluated when the pattern matches.

`patterndb2,3.xml`

```

<?xml version='1.0' encoding='UTF-8'?>
<patterndb>
  <created>2018-01-01 01:02:03</created>
  <version>4</version>

  <group>
    <name>ssh</name>
    <id>1</id>
    <matchfield>
      <name>SourceName</name>
      <type>exact</type>
      <value>sshd</value>
    </matchfield>

    <pattern>
      <id>1</id>
      <name>ssh auth success</name>

      <matchfield>
        <name>Message</name>
        <type>regexp</type>
        <value>^Accepted (\S+) for (\S+) from (\S+) port \d+ ssh2</value>
        <capturedfield>
          <name>AuthMethod</name>
          <type>string</type>
        </capturedfield>
        <capturedfield>
```

```
<name>AccountName</name>
<type>string</type>
</capturedfield>
<capturedfield>
<name>SourceIP4Address</name>
<type>ip4addr</type>
</capturedfield>
</matchfield>

<set>
<field>
<name>TaxonomyStatus</name>
<value>success</value>
<type>string</type>
</field>
<field>
<name>TaxonomyAction</name>
<value>authenticate</value>
<type>string</type>
</field>
</set>
</pattern>

<pattern>
<id>2</id>
<name>ssh auth failure</name>

<matchfield>
<name>Message</name>
<type>regexp</type>
<value>^Failed (\S+) for invalid user (\S+) from (\S+) port \d+ ssh2</value>

<capturedfield>
<name>AuthMethod</name>
<type>string</type>
</capturedfield>
<capturedfield>
<name>AccountName</name>
<type>string</type>
</capturedfield>
<capturedfield>
<name>SourceIP4Address</name>
<type>ip4addr</type>
</capturedfield>
</matchfield>

<set>
<field>
<name>TaxonomyStatus</name>
<value>failure</value>
<type>string</type>
</field>
<field>
<name>TaxonomyAction</name>
<value>authenticate</value>
<type>string</type>
</field>
</set>

<exec>
```

```
$TestField = 'test';
$TestField = $Testfield + 'value';
</exec>
</pattern>

</group>

</patterndb>
```

Example 415. Using the `match_pattern()` Function

This example is the same as the previous one, and uses the same [pattern file](#), but it uses the [`match_pattern\(\)`](#) function to discard any event that is not matched by the pattern file.

nxlog.conf

```
1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension pattern>
6   Module      xm_pattern
7   PatternFile modules/extension/pattern/patterndb2,3.xml
8 </Extension>
9
10 <Input in>
11   Module     im_file
12   File       'test2.log'
13   <Exec>
14     parse_syslog();
15     if not match_pattern() drop();
16   </Exec>
17 </Input>
```

86.21. Perl (xm_perl)

The [Perl programming language](#) is widely used for log processing and comes with a broad set of modules bundled or available from [CPAN](#). Code can be written more quickly in Perl than in C, and code execution is safer because exceptions (`croak/die`) are handled properly and will only result in an unfinished attempt at log processing rather than taking down the whole NXLog process.

While the [NXLog language](#) is already a powerful framework, it is not intended to be a fully featured programming language and does not provide lists, arrays, hashes and other features available in many high-level languages. With this module, Perl can be used to process event data via a built-in Perl interpreter. See also the [`im_perl`](#) and [`om_perl`](#) modules.

The Perl interpreter is only loaded if the module is declared in the configuration. The module will parse the file specified in the `PerlCode` directive when NXLog starts the module. This file should contain one or more methods which can be called from the `Exec` directive of any module that will use Perl for log processing. See the [example](#) below.

To access event data, the `Log::Nxlog` module must be included, which provides the following methods.

log_debug(msg)

Send the message `msg` to the internal logger on DEBUG log level. This method does the same as the [`log_debug\(\)`](#) procedure in NXLog.

`log_info(msg)`

Send the message *msg* to the internal logger on INFO log level. This method does the same as the [log_info\(\)](#) procedure in NXLog.

`log_warning(msg)`

Send the message *msg* to the internal logger on WARNING log level. This method does the same as the [log_warning\(\)](#) procedure in NXLog.

`log_error(msg)`

Send the message *msg* to the internal logger on ERROR log level. This method does the same as the [log_error\(\)](#) procedure in NXLog.

`delete_field(event, key)`

Delete the value associated with the field named *key*.

`field_names(event)`

Return a list of the field names contained in the event data. This method can be used to iterate over all of the fields.

`field_type(event, key)`

Return a string representing the type of the value associated with the field named *key*.

`get_field(event, key)`

Retrieve the value associated with the field named *key*. This method returns a scalar value if the key exists and the value is defined, otherwise it returns undef.

`set_field_boolean(event, key, value)`

Set the boolean value in the field named *key*.

`set_field_integer(event, key, value)`

Set the integer value in the field named *key*.

`set_field_string(event, key, value)`

Set the string value in the field named *key*.

For the full NXLog Perl API, see the POD documentation in [Nxlog.pm](#). The documentation can be read with `perldoc Log::Nxlog`.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.21.1. Configuration

The `xm_perl` module accepts the following directives in addition to the [common module directives](#).

PerlCode

This mandatory directive expects a file containing valid Perl code. This file is read and parsed by the Perl interpreter. Methods defined in this file can be called with the [call\(\)](#) procedure.

86.21.2. Procedures

The following procedures are exported by `xm_perl`.

`call(string subroutine);`

Call the given Perl *subroutine*.

```
perl_call(string subroutine, varargs args);
```

Call the given Perl *subroutine*.

86.21.3. Examples

Example 416. Using the built-in Perl interpreter

In this example, logs are parsed as Syslog and then are passed to a Perl method which does a GeolP lookup on the source address of the incoming message.

nxlog.conf

```
1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension perl>
6   Module      xm_perl
7   PerlCode   modules/extension/perl/processlogs.pl
8 </Extension>
9
10 <Output fileout>
11   Module     om_file
12   File       'tmp/output'
13
14   # First we parse the input natively from nxlog
15   Exec      parse_syslog_bsd();
16
17   # Now call the 'process' subroutine defined in 'processlogs.pl'
18   Exec      perl_call("process");
19
20   # You can also invoke this public procedure 'call' in case
21   # of multiple xm_perl instances like this:
22   # Exec      perl->call("process");
23 </Output>
```

```
processlogs.pl

use strict;
use warnings;

# Without Log::Nxlog you cannot access (read or modify) the event data
use Log::Nxlog;

use Geo::IP;

my $geoip;

BEGIN
{
    # This will be called once when nxlog starts so you can use this to
    # initialize stuff here
    $geoip = Geo::IP->new(GEOIP_MEMORY_CACHE);
}

# This is the method which is invoked from 'Exec' for each event
sub process
{
    # The event data is passed here when this method is invoked by the module
    my ( $event ) = @_;

    # We look up the country of the sender of the message
    my $msgsrcaddr = Log::Nxlog::get_field($event, 'MessageSourceAddress');
    if ( defined($msgsrcaddr) )
    {
        my $country = $geoip->country_code_by_addr($msgsrcaddr);
        $country = "unknown" unless ( defined($country) );
        Log::Nxlog::set_field_string($event, 'MessageSourceCountry', $country);
    }

    # Iterate over the fields
    foreach my $fname ( @{$Log::Nxlog::field_names($event)} )
    {
        # Delete all fields except these
        if ( ! ( ($fname eq 'raw_event') ||
                  ($fname eq 'AccountName') ||
                  ($fname eq 'MessageSourceCountry') ) )
        {
            Log::Nxlog::delete_field($event, $fname);
        }
    }

    # Check a field and rename it if it matches
    my $accountname = Log::Nxlog::get_field($event, 'AccountName');
    if ( defined($accountname) && ($accountname eq 'John') )
    {
        Log::Nxlog::set_field_string($event, 'AccountName', 'johnny');
        Log::Nxlog::log_info('renamed john');
    }
}
```

86.22. Python (xm_python)

This module provides support for processing NXLog log data with methods written in the [Python](#) language. The file specified by the [PythonCode](#) directive should contain one or more methods which can be called from the

Exec directive of any module. See also the [im_python](#) and [om_python](#) modules.

The Python script should import the [nxlog](#) module, and will have access to the following classes and functions.

`nxlog.log_debug(msg)`

Send the message *msg* to the internal logger at DEBUG log level. This function does the same as the core [log_debug\(\)](#) procedure.

`nxlog.log_info(msg)`

Send the message *msg* to the internal logger at INFO log level. This function does the same as the core [log_info\(\)](#) procedure.

`nxlog.log_warning(msg)`

Send the message *msg* to the internal logger at WARNING log level. This function does the same as the core [log_warning\(\)](#) procedure.

`nxlog.log_error(msg)`

Send the message *msg* to the internal logger at ERROR log level. This function does the same as the core [log_error\(\)](#) procedure.

`class nxlog.Module()`

This class is instantiated by NXLog and can be accessed via the [LogData\(\).module](#) attribute. This can be used to set or access variables associated with the module (see the [example](#) below).

`class nxlog.LogData()`

This class represents an event. It is instantiated by NXLog and passed to the method specified by the [python_call\(\)](#) procedure.

`delete_field(name)`

This method removes the field *name* from the event record.

`field_names()`

This method returns a list with the names of all the fields currently in the event record.

`get_field(name)`

This method returns the value of the field *name* in the event.

`set_field(name, value)`

This method sets the value of field *name* to *value*.

`module`

This attribute is set to the Module object associated with the event.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.22.1. Configuration

The *xm_python* module accepts the following directives in addition to the [common module directives](#).

`PythonCode`

This mandatory directive specifies a file containing Python code. The [python_call\(\)](#) procedure can be used to call a Python function defined in the file. The function must accept an [nxlog.LogData\(\)](#) object as its argument.

86.22.2. Procedures

The following procedures are exported by xm_python.

`python_call(string function);`

Call the specified *function*, which must accept an `nxlog.LogData()` object as its only argument.

86.22.3. Examples

Example 417. Using Python for Log Processing

This configuration calls two Python functions to modify each event record. The `add_checksum()` uses Python's `hashlib` module to add a `$ChecksumSHA1` field to the event; the `add_counter()` function adds a `$Counter` field for non-DEBUG events.

NOTE

The `pm_hmac` module offers a more complete implementation for checksumming. See [Statistical Counters](#) for a native way to add counters.

`nxlog.conf`

```
1 <Extension _json>
2   Module      xm_json
3   DateFormat YYYY-MM-DD hh:mm:ss
4 </Extension>
5
6 <Extension _syslog>
7   Module      xm_syslog
8 </Extension>
9
10 <Extension python>
11  Module      xm_python
12  PythonCode  modules/extension/python/processlogs2.py
13 </Extension>
14
15 <Output out>
16  Module      om_file
17  File        'tmp/output'
18  <Exec>
19    # The $SeverityValue field is added by this procedure.
20    # Most other parsers also add a normalized severity value.
21    parse_syslog();
22
23    # Add a counter for each event with log level above DEBUG.
24    python_call('add_counter');
25
26    # Calculate a checksum (after the counter field is added).
27    python_call('add_checksum');
28
29    # Convert to JSON format
30    to_json();
31  </Exec>
32 </Output>
```

```
processlogs2.py

import hashlib

import nxlog

def add_checksum(event):
    # Convert field list to dictionary
    all = {}
    for field in event.field_names():
        all.update({field: event.get_field(field)})

    # Calculate checksum and add to event record
    checksum = hashlib.sha1(repr(sorted(all.items()))).hexdigest()
    event.set_field('ChecksumSHA1', checksum)
    nxlog.log_debug('Added checksum field')

def add_counter(event):
    # Get module object and initialize counter
    module = event.module
    if not 'counter' in module:
        module['counter'] = 0
        nxlog.log_debug('Initialized counter field')

    # Skip DEBUG messages
    severity = event.get_field('SeverityValue')
    if severity > 1:
        # Add field
        event.set_field('Counter', module['counter'])
        nxlog.log_debug('Added counter field')

    # Increment counter
    module['counter'] += 1
    nxlog.log_debug('Incremented counter')
```

86.23. Resolver (xm_resolver)

This module provides provides functions for resolving (converting between) IP addresses and names, and between group/user ids and names. The module uses an internal cache in order to minimize the number of DNS lookup queries.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.23.1. Configuration

The *xm_resolver* module accepts the following directives in addition to the [common module directives](#).

CacheExpiry

Specifies the time in seconds after entries in the cache are considered invalid and are refreshed by issuing a DNS lookup. The default expiry is 3600 seconds.

CacheLimit

This directive can be used to specify an upper limit on the number of entries in the cache in order to prevent the cache from becoming arbitrary large and potentially exhausting memory. When the number of entries in the cache reaches this value, no more items will be inserted into the cache. The default is 100,000 entries.

86.23.2. Functions

The following functions are exported by xm_resolver.

string gid_to_name(integer gid)

Return the group name assigned to the group ID. If *gid* cannot be looked up, undef is returned.

string gid_to_name(string gid)

Return the group name assigned to the string *gid* on Unix. If *gid* cannot be looked up, undef is returned.

integer group_get_gid(string groupname)

Return the group ID assigned to the group name.

string ipaddr_to_name(unknown ipaddr)

Resolve and return the DNS name assigned to the IP address. The *ipaddr* argument can be either a **String** or an **ip4addr** type.

ip4addr name_to_ipaddr(string name)

Resolve and return the first IPv4 address assigned to *name*.

string uid_to_name(integer uid)

Return the username corresponding to the user ID. If *uid* cannot be looked up, undef is returned.

string uid_to_name(string uid)

Return the username corresponding to the user ID or SID. This function takes a string which is normally a SID on Windows or an integer UID on Unix. On Windows this function will convert the SID to a string in the format of **DOMAIN\USER**. If *uid* cannot be looked up, undef is returned.

integer user_get_gid(string username)

Return the user's group ID (the group ID assigned to *username*).

integer user_get_uid(string username)

Return the user ID assigned to *username*.

86.23.3. Examples

Example 418. Using Functions Provided by xm_resolver

It is common for devices to send Syslog messages containing the IP address of the device instead of a real hostname. In this example, Syslog messages are parsed and the hostname field of each Syslog header is converted to a hostname if it looks like an IP address.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension _resolver>
6   Module      xm_resolver
7 </Extension>
8
9 <Input tcp>
10  Module     im_tcp
11  Host       0.0.0.0
12  Port       1514
13  <Exec>
14    parse_syslog();
15    if $Hostname =~ /\d+\.\d+\.\d+\.\d+/
16    {
17      $HostIP = $Hostname;
18      $Hostname = ipaddr_to_name($HostIP);
19      if not defined $Hostname $Hostname = $HostIP;
20    }
21  </Exec>
22 </Input>
23
24 <Output file>
25   Module     om_file
26   File       'tmp/output'
27   Exec       to_syslog_bsd();
28 </Output>
29
30 <Route tcp_to_file>
31   Path       tcp => file
32 </Route>
```

Input Sample

```

<38>2014-11-11 11:40:27 127.0.0.1 sshd[3436]: Failed none for invalid user asdf from 127.0.0.1
port 51824 ssh2<
<38>2014-11-12 12:42:37 127.0.0.1 sshd[3436]: Failed password for invalid user fdsa from
127.0.0.1 port 51824 ssh2<
```

Output Sample

```

<38>Nov 11 11:40:27 localhost sshd[3436]: Failed none for invalid user asdf from 127.0.0.1 port
51824 ssh2<
<38>Nov 12 12:42:37 localhost sshd[3436]: Failed password for invalid user fdsa from 127.0.0.1
port 51824 ssh2<
```

86.24. Rewrite (xm_rewrite)

This module can be used to transform event records by:

- renaming fields,

- deleting specified fields (blacklist),
- keeping only a list of specified fields (whitelist), and
- evaluating additional statements.

The `xm_rewrite` module provides [Delete](#), [Keep](#), and [Rename](#) directives for modifying event records. With the [Exec](#) directive of this module, it is possible to invoke functions and procedures from other modules. This allows all data transformation to be configured in a single module instance in order to simplify the configuration. Then the transformation can be referenced from another module by adding:

```
Exec rewrite->process();
```

This same statement can be used by more than one module instance if necessary, rather than duplicating configuration.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.24.1. Configuration

The `xm_rewrite` module accepts the following directives in addition to the [common module directives](#).

The order of the action directives is significant as the module executes them in the order of appearance.

Delete

This directive takes a field name or a list of fields. The fields specified will be removed from the event record. This can be used to blacklist specific fields that are not wanted in the event record. This is equivalent to using [delete\(\)](#) in [Exec](#).

Exec

This directive does the same as the [Exec](#) directive in other modules: the statement provided in the argument will be evaluated.

Keep

This directive takes a field name or a list of fields. The fields specified will be kept and all other fields not appearing in the list will be removed from the event record. This can be used to whitelist specific fields.

Rename

This directive takes two fields. The field in the first argument will be renamed to the name in the second. This is equivalent to using [rename_field\(\)](#) in [Exec](#).

86.24.2. Procedures

The following procedures are exported by `xm_rewrite`.

```
process();
```

This procedure invokes the data processing as specified in the configuration of the `xm_rewrite` module instance.

86.24.3. Examples

Example 419. Using xm_rewrite to Transform Syslog Data Read from File

The following configuration parses Syslog data from a file, invokes the `process()` procedure of the `xm_rewrite` instance to keep and rename whitelisted fields, then writes JSON-formatted output to a file.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension rewrite>
6   Module      xm_rewrite
7   Keep        EventTime, Severity, Hostname, SourceName, Message
8   Rename     EventTime, timestamp
9   Rename     Hostname, host
10  Rename    SourceName, src
11  Rename    Message, msg
12  Rename    Severity, sev
13  Exec      if $msg =~ /error/ $sev = 'ERROR';
14 </Extension>
15
16 <Extension json>
17   Module      xm_json
18 </Extension>
19
20 <Input syslogfile>
21   Module      im_file
22   File       "modules/extension/rewrite/xm_rewrite.in"
23   Exec      parse_syslog();
24   Exec      rewrite->process();
25 </Input>
26
27 <Output fileout>
28   Module      om_file
29   File       'tmp/output'
30   Exec      to_json();
31 </Output>
32
33 <Route rewrite_fields>
34   Path      syslogfile => fileout
35 </Route>
```

Input Sample

```
<0>2010-10-12 12:49:06 mybox app[12345]: kernel message<
<30>2010-10-12 12:49:06 mybox app[12345]: daemon - info<
<27>2010-10-12 12:49:06 mybox app[12345]: daemon - error<
<30>2010-10-12 13:19:11 mybox app[12345]: There was an error<
```

Output Sample

```
{"sev": "CRITICAL", "host": "mybox", "timestamp": "2010-10-12 12:49:06", "src": "app", "msg": "kernel message"}
{"sev": "INFO", "host": "mybox", "timestamp": "2010-10-12 12:49:06", "src": "app", "msg": "daemon - info"}
{"sev": "ERROR", "host": "mybox", "timestamp": "2010-10-12 12:49:06", "src": "app", "msg": "daemon - error"}
{"sev": "ERROR", "host": "mybox", "timestamp": "2010-10-12 13:19:11", "src": "app", "msg": "There was an error"}
```

Example 420. Performing Additional Parsing in an xm_rewrite Module Instance

The following configuration does the exact same processing. In this case, however, the Syslog parsing is moved into the `xm_rewrite` module instance so the input module only needs to invoke the `process()` procedure.

`nxlog.conf`

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension rewrite>
6   Module      xm_rewrite
7   Exec        parse_syslog();
8   Keep       EventTime, Severity, Hostname, SourceName, Message
9   Rename     EventTime, timestamp
10  Rename    Hostname, host
11  Rename    SourceName, src
12  Rename    Message, msg
13  Rename    Severity, sev
14  Exec      if $msg =~ /error/ $sev = 'ERROR';
15 </Extension>
16
17 <Extension json>
18   Module      xm_json
19 </Extension>
20
21 <Input syslogfile>
22   Module      im_file
23   File        "modules/extension/rewrite/xm_rewrite.in"
24   Exec        rewrite->process();
25 </Input>
26
27 <Output fileout>
28   Module      om_file
29   File        'tmp/output'
30   Exec        to_json();
31 </Output>
32
33 <Route rewrite_fields>
34   Path        syslogfile => fileout
35 </Route>
```

86.25. Ruby (xm_ruby)

This module provides support for processing NXLog log data with methods written in the [Ruby](#) language. Ruby methods can be defined in a script and then called from the `Exec` directive of any module that will use Ruby for log processing. See the [example](#) below. See also the `im_ruby` and `om_ruby` modules.

The `Nxlog` module provides the following classes and methods.

`Nxlog.log_info(msg)`

Send the message `msg` to the internal logger at DEBUG log level. This method does the same as the core `log_debug()` procedure.

`Nxlog.log_debug(msg)`

Send the message `msg` to the internal logger at INFO log level. This method does the same as the core

[log_info\(\)](#) procedure.

Nxlog.log_warning(msg)

Send the message *msg* to the internal logger at WARNING log level. This method does the same as the core [log_warning\(\)](#) procedure.

Nxlog.log_error(msg)

Send the message *msg* to the internal logger at ERROR log level. This method does the same as the core [log_error\(\)](#) procedure.

class Nxlog.LogData

This class represents an event.

field_names()

This method returns an array with the names of all the fields currently in the event record.

get_field(name)

This method returns the value of the field *name* in the event.

set_field(name, value)

This method sets the value of field *name* to *value*.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.25.1. Configuration

The *xm_ruby* module accepts the following directives in addition to the [common module directives](#).

RubyCode

This mandatory directive expects a file containing valid Ruby code. Methods defined in this file can be called with the [ruby_call\(\)](#) procedure.

86.25.2. Procedures

The following procedures are exported by *xm_ruby*.

call(string subroutine);

Calls the Ruby method provided in the first argument.

ruby_call(string subroutine);

Calls the Ruby method provided in the first argument.

86.25.3. Examples

Example 421. Processing Logs With Ruby

In this example logs are parsed as Syslog, then the data is passed to a Ruby method which adds an incrementing **\$AlertCounter** field for any event with a normalized **\$SeverityValue** of at least 4.

nxlog.conf

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Extension ruby>
6   Module      xm_ruby
7   RubyCode    ./modules/extension/ruby/processlogs2.rb
8 </Extension>
9
10 <Input in>
11   Module     im_file
12   File       'test2.log'
13   <Exec>
14     parse_syslog();
15     ruby->call('add_alert_counter');
16   </Exec>
17 </Input>
```

processlogs2.rb

```

$counter = 0

def add_alert_counter(event)
  if event.get_field('SeverityValue') >= 4
    Nxlog.log_debug('Adding AlertCounter field')
    $counter += 1
    event.set_field('AlertCounter', $counter)
  end
end
```

86.26. SNMP Traps (xm_snmp)

This module provides support for parsing SNMP v1, v2c, and v3 trap messages. For SNMP v3, the user-based security model (USM) is supported, providing both authentication and encryption functionality. Instead of parsing log files or piping in input from snmptrapd, this module provides convenient and efficient trap variables easily accessible in NXLog fields. There is no need to manually parse the output of external tools, thus it provides a better all-in-one solution for SNMP trap reception.

Like the [xm_syslog](#) module, the [xm_snmp](#) module does not provide support for the network transport layer. Since traps are sent primarily over UDP (typically to port 162), the [im_udp](#) module should be used together with this module. This module registers an input reader function under the name "snmp" which can be used in the [InputType](#) directive to parse UDP message payloads.

The module supports MIB definitions in order to resolve OID numbers to names. In addition to the standard [xm_snmp](#) module fields and the [im_udp](#) module fields, each variable in the trap message will be available as an internal NXLog field. If the OID cannot be resolved to a name, the string **OID**. will be prepended to the dotted OID number representation in the field name. For example, if a trap contains a string variable with OID **1.3.6.1.4.1.311.1.13.1.9999.3.0**, this field can be accessed as the NXLog field **\$OID.1.3.6.1.4.1.311.1.13.1.9999.3.0**. If the object identifier can be resolved to a name called **FIELDNAME**, the value will be available in the NXLog field **\$SNMP.FIELDNAME**. The SNMP trap variables are also put in the **\$raw_event** field and are listed as **name="value"** pairs there in the order they appear in the trap. The following is an example of the contents of the **\$raw_event** field (line breaks added):

```
2011-12-15 18:10:35 192.1.1.114 INFO \
OID.1.3.6.1.4.1.311.1.13.1.9999.1.0="test msg" \
OID.1.3.6.1.4.1.311.1.13.1.9999.2.0="Administrator" \
OID.1.3.6.1.4.1.311.1.13.1.9999.3.0="WIN-OUNNPISDHIG" \
OID.1.3.6.1.4.1.311.1.13.1.9999.4.0="1" \
OID.1.3.6.1.4.1.311.1.13.1.9999.5.0="0" \
OID.1.3.6.1.4.1.311.1.13.1.9999.6.0="test msg"
```

NOTE To convert the output to Syslog format, consider using one of the [to_syslog\(\)](#) procedures provided by the [xm_syslog](#) module. However, note that the resulting format will not be in accordance with [RFC 5675](#).

Microsoft Windows can convert and forward EventLog messages as SNMPv1 traps. The *evntwin* utility can be used to configure which events are sent as traps. See [How to Generate SNMP traps from Windows Events](#) for more information about setting up this feature.

The [Net-SNMP](#) toolkit (available for Unix/Linux and Windows) provides the *snmptrap* command line utility which can be used for sending test SNMP traps. Create the following MIB definition file and put it in a directory specified by the [MIBDir](#) directive:

MIB Definition File

```
TRAP-TEST-MIB DEFINITIONS ::= BEGIN
IMPORTS ucdExperimental FROM UCD-SNMP-MIB;

demotrap OBJECT IDENTIFIER ::= { ucdExperimental 990 }

demo-trap TRAP-TYPE
STATUS current
ENTERPRISE demotrap
VARIABLES { sysLocation }
DESCRIPTION "This is just a demo"
 ::= 17

END
```

Here is an example for invoking the *snmptrap* utility (line break added):

```
snmptrap -v 1 -c public localhost TRAP-TEST-MIB::demotrap \
    "" 6 17 "" sysLocation s "Test message"
```

The received trap should look like this in the [\\$raw_event](#) field:

```
2011-12-15 18:21:46 192.168.168.2 INFO SNMP.sysLocation="Test message"
```

If the MIB definition can not be loaded or parsed, the unresolved OID number will be seen in the message:

```
2011-12-15 19:43:54 192.168.168.2 INFO OID.1.3.6.1.2.1.1.6="Test message"
```

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.26.1. Configuration

The *xm_snmp* module accepts the following directives in addition to the [common module directives](#).

AllowAuthenticatedOnly

This boolean directive specifies whether only authenticated SNMP v3 traps should be accepted. If set to TRUE, the [User](#) block must also be defined, and unauthenticated SNMP traps are not accepted. The default is FALSE: all SNMP traps are accepted.

MIBDir

This optional directive can be used to define a directory which contains MIB definition files. Multiple **MIBDir** directives can be specified.

User

This directive is specified as a block (see [Parsing Authenticated and Encrypted SNMP Traps](#)) and provides the authentication details for an SNMP v3 user. The block must be named with the corresponding user. This block can be specified more than once to provide authentication details for multiple users.

AuthPasswd

This required directive specifies the authentication password.

AuthProto

This optional directive specifies the authentication protocol to use. Supported values are **md5** and **sha1**. If this directive is not specified, the default is **md5**.

EncryptPasswd

This directive specifies the encryption password to use for encrypted traps.

EncryptProto

This optional directive specifies the encryption protocol to use. Supported values are **des** and **aes**. The default, if encryption is in use and this directive is not specified, is **des**.

86.26.2. Fields

The following fields are used by xm_snmp.

\$raw_event (type: string)

For SNMP v1, a string containing the **\$EventTime**, **\$SNMP.MessageSourceAddress**, **\$Severity**, and **\$SNMP.TrapNameGeneric** fields and a list of key-value pairs. For SNMP v2c and v3, a string containing the **\$EventTime** and **\$Severity** fields and a list of key-value pairs.

\$EventTime (type: datetime)

The reception time of the trap.

\$Severity (type: string)

The severity name: **INFO** (there is no severity in SNMP traps).

\$SeverityValue (type: integer)

The INFO severity level value: **2** (because there is no severity in SNMP traps).

\$SNMP . CommunityString (type: string)

The community string within the SNMP message.

\$SNMP . MessageSourceAddress (type: string)

The IP address of the sender as provided in the trap message. Note that there is a **\$MessageSourceAddress** field set by the **im_udp** module. Available in SNMP v1 only.

\$SNMP . RequestID (type: integer)

An integer associating the SNMP response with a particular SNMP request.

\$SNMP . TrapCodeGeneric (type: integer)

Indicates one of a number of generic trap types. Available in SNMP v1 only.

\$SNMP.TrapCodeSpecific (type: *integer*)

A code value indicating an implementation-specific trap type. Available in SNMP v1 only.

\$SNMP.TrapName (type: *string*)

The resolved name of the object identifier in [SNMP.TrapOID](#). The field will be unset if the OID cannot be resolved. Available in SNMP v1 only.

\$SNMP.TrapNameGeneric (type: *string*)

The textual representation of [SNMP.TrapCodeGeneric](#), one of: `coldStart(0)`, `warmStart(1)`, `linkDown(2)`, `linkUp(3)`, `authenticationFailure(4)`, `egpNeighborLoss(5)`, or `enterpriseSpecific(6)`. Available in SNMP v1 only.

\$SNMP.TrapOID (type: *string*)

The object identifier of the TRAP message. Available in SNMP v1 only.

\$sysUptime (type: *integer*)

The amount of time that has elapsed between the last network reinitialization and generation of the trap. This name is chosen in accordance with RFC 5424. Available in SNMP v1 only.

86.26.3. Examples

Example 422. Using MIB Definitions to Parse SNMP Traps

The `InputType snmp` directive in the `im_udp` module block is required to parse the SNMP payload in the UDP message.

nxlog.conf

```
1 <Extension snmp>
2   Module      xm_snmp
3   MIBDir     /usr/share/mibs/iana
4   MIBDir     /usr/share/mibs/ietf
5   MIBDir     /usr/share/mibs/site
6 </Extension>
7
8 <Input udp>
9   Module      im_udp
10  Host        0.0.0.0
11  Port        162
12  InputType   snmp
13 </Input>
```

Example 423. Parsing Authenticated and Encrypted SNMP Traps

This configuration parses SNMP v3 traps. Only authenticated traps are parsed; a warning is printed for each non-authenticated source that sends a trap. The **User** block provides authentication and encryption settings for the **switch1** user.

nxlog.conf

```

1 <Extension snmp>
2   Module           xm_snmp
3   MIBDir          /usr/share/mibs/iana
4   MIBDir          /usr/share/mibs/ietf
5   AllowAuthenticatedOnly TRUE
6   <User switch1>
7     AuthPasswd    secret
8     AuthProto      sha1
9     EncryptPasswd secret
10    EncryptProto   aes
11  </User>
12 </Extension>
13
14 <Input udp>
15   Module      im_udp
16   Host        0.0.0.0
17   Port        162
18   InputType   snmp
19 </Input>
```

86.27. Remote Management (xm_soapadmin)

This module provides secure remote administration capabilities to the NXLog engine using SOAP over HTTP/HTTPS (also known as web services). SOAP is a widespread protocol and can be used from many different programming languages, consequently it is easy to implement administration scripts or create plugins for system monitoring tools such as Nagios, Munin or Cacti. Using the *xm_soapadmin* module, NXLog can accept and initiate connections over TCP, SSL, and Unix domain sockets depending on its [configuration](#). This module is superseded by [xm_admin](#) and will eventually be phased out. The [xm_admin](#) can be used as a direct replacement of this module for all intents and purposes.

Note that though the module can both initiate and accept connections, the direction of the HTTP requests is always the same: requests are sent to the module and it returns HTTP responses.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.27.1. Configuration

The *xm_soapadmin* module accepts the following directives in addition to the [common module directives](#).

Connect

This directive instructs the module to connect to a remote socket. The argument must be an IP address when **SocketType** is set to [TCP](#) or [SSL](#). Otherwise it must be a name of a socket for [UDS](#). **Connect** cannot be used together with the [Listen](#) directive. Multiple *xm_soapadmin* instances may be configured if multiple administration ports are required.

Listen

This directive instructs the module to accept connections. The argument must be an IP address when **SocketType** is [TCP](#) or [SSL](#). Otherwise it must be the name of a socket for [UDS](#). **Listen** cannot be used together with the [Connect](#) directive. Multiple *xm_soapadmin* instances may be configured if multiple administration

ports are required. If neither **Listen** nor **Connect** are specified, the module will listen with **SocketType TCP** on **127.0.0.1**.

Port

This specifies the port number used with the **Listen** or **Connect** modes. The default port is **8080**.

ACL

This block defines directories which can be used with the **GetFile** and **PutFile** web service requests. The name of the ACL is used in these requests together with the filename. The filename can contain only characters **[a-zA-Z0-9-_]**, so these file operations will only work within the directory.

AllowRead

If set to TRUE, **GetFile** requests are allowed.

AllowWrite

If set to TRUE, **PutFile** requests are allowed.

Directory

The name of the directory where the files are saved to or loaded from.

Example 424. ACL Block Allowing Read and Write on Files in the Directory

This ACL is named "conf" and allows both **GetFile** and **PutFile** requests on the specified directory.

```
<ACL conf>
    Directory /var/run/nxlog/configs
    AllowRead TRUE
    AllowWrite TRUE
</ACL>
```

AllowIP

This optional directive can be used to specify an IP address or a network that is allowed to connect. The directive can be specified more than once to add different IPs or networks to the whitelist. The following formats can be used:

- **0.0.0.0** (IPv4 address)
- **0.0.0.0/32** (IPv4 network with subnet bits)
- **0.0.0.0/0.0.0.0** (IPv4 network with subnet address)
- **aa::1** (IPv6 address)
- **aa::12/64** (IPv6 network with subnet bits)

AllowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with unknown and self-signed certificates. The default value is FALSE: all connections must present a trusted certificate. This directive is only valid if **SocketType** is set to **SSL**.

CADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format. This directive is only valid if **SocketType** is set to **SSL**.

CAFfile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote socket. This directive is only valid if [SocketType](#) is set to [SSL](#).

CertFile

This specifies the path of the certificate file to be used for SSL connections. This directive is only valid if [SocketType](#) is set to [SSL](#).

CertKeyFile

This specifies the path of the certificate key file to be used for SSL connections. This directive is only valid if [SocketType](#) is set to [SSL](#).

CRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format. This directive is only valid if [SocketType](#) is set to [SSL](#).

CRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote socket. This directive is only valid if [SocketType](#) is set to [SSL](#).

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [CertKeyFile](#). This directive is only valid if [SocketType](#) is set to [SSL](#). This directive is not needed for password-less private keys.

Reconnect

This directive has been deprecated as of version 2.4. The module will try to reconnect automatically at increasing intervals on all errors.

RequireCert

This boolean directive specifies that the remote must present a certificate. If set to TRUE and there is no certificate presented during the connection handshake, the connection will be refused. The default value is TRUE: each connection must use a certificate. This directive is only valid if [SocketType](#) is set to [SSL](#).

SocketType

This directive sets the connection type. It can be one of the following:

SSL

TLS/SSL for secure network connections

TCP

TCP, the default if [SocketType](#) is not explicitly specified

UDS

Unix domain socket

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: [SSLv2](#), [SSLv3](#), [TLSv1](#), [TLSv1.1](#), and [TLSv1.2](#). By default, the [SSLv3](#) and [TLSv1.2](#) protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

86.27.2. Exported SOAP Methods

The *xm_soapadmin* module exports the following SOAP methods (web services) which can be called remotely. There is a WSDL file which can be used by different developer tools to easily hook into the exported WS API to reduce development time.

GetFile

Download a file from the NXLog agent. This will only work if the specified [ACL](#) exists.

ModuleInfo

Request information about a module instance.

ModuleRestart

Restart a module instance.

ModuleStart

Start a module instance.

ModuleStop

Stop a module instance.

PutFile

Upload a file to the NXLog agent. This will only work if the specified [ACL](#) exists. A file can be a configuration file, certificate or certificate key, pattern database, correlation rule file, etc. Using this method enables NXLog to be reconfigured from a remote host.

ServerInfo

Request information about the server. This will also return info about all module instances.

ServerRestart

Restart the server.

ServerStart

Start all modules of the server, the opposite of [*ServerStop*](#).

ServerStop

Stop all modules of the server. Note that the NXLog process will not exit, otherwise it would be impossible to make it come back online remotely. Extension modules are not stopped for the same reason.

86.27.3. Examples

Example 425. Configuration for Multiple Admin Ports

This configuration specifies two additional administration ports on localhost.

nxlog.conf

```

1 <Extension ssl_connect>
2   Module      xm_soapadmin
3   Connect     192.168.1.1
4   Port        4041
5   SocketType  SSL
6   CAFile      %CERTDIR%/ca.pem
7   CertFile    %CERTDIR%/client-cert.pem
8   CertKeyFile %CERTDIR%/client-key.pem
9   KeyPass     secret
10  AllowUntrusted FALSE
11  RequireCert  TRUE
12  Reconnect    60
13  <ACL conf>
14    Directory  %CONFDIR%
15    AllowRead  TRUE
16    AllowWrite TRUE
17  </ACL>
18  <ACL cert>
19    Directory  %CERTDIR%
20    AllowRead  TRUE
21    AllowWrite TRUE
22  </ACL>
23 </Extension>
24
25 <Extension tcp_listen>
26   Module      xm_soapadmin
27   Listen      localhost
28   Port        8080
29 </Extension>
30
31 <Extension tcp_connect>
32   Module      xm_soapadmin
33   Connect     localhost
34   Port        4040
35 </Extension>
```

86.28. Syslog (xm_syslog)

This module provides support for the legacy BSD Syslog protocol as defined in RFC 3164 and the current IETF standard defined by RFCs 5424-5426. This is achieved by exporting functions and procedures usable from the NXLog language. The transport is handled by the respective input and output modules (such as [im_udp](#)), this module only provides a parser and helper functions to create Syslog messages and handle facility and severity values.

The older but still widespread BSD Syslog standard defines both the format and the transport protocol in RFC 3164. The transport protocol is UDP, but to provide reliability and security, this line-based format is also commonly transferred over TCP and SSL. There is a newer standard defined in RFC 5424, also known as the IETF Syslog format, which obsoletes the BSD Syslog format. This format overcomes most of the limitations of BSD Syslog and allows multi-line messages and proper timestamps. The transport method is defined in RFC 5426 for UDP and RFC 5425 for TLS/SSL.

Because the IETF Syslog format supports multi-line messages, RFC 5425 defines a special format to encapsulate these by prepending the payload size in ASCII to the IETF Syslog message. Messages transferred in UDP packets

are self-contained and do not need this additional framing. The following input reader and output writer functions are provided by the `xm_syslog` module to support this TLS transport defined in RFC 5425. While RFC 5425 explicitly defines that the TLS network transport protocol is to be used, pure TCP may be used if security is not a requirement. Syslog messages can also be written to file with this framing format using these functions.

InputType Syslog_TLS

This input reader function parses the payload size and then reads the message according to this value. It is required to support Syslog TLS transport defined in RFC 5425.

OutputType Syslog_TLS

This output writer function prepends the payload size to the message. It is required to support Syslog TLS transport defined in RFC 5425.

NOTE The `Syslog_TLS` InputType/OutputType can work with any input/output such as `im_tcp` or `im_file` and does not depend on SSL transport at all. The name `Syslog_TLS` was chosen to refer to the octet-framing method described in RFC 5425 used for TLS transport.

NOTE The `pm_transformer` module can also parse and create BSD and IETF Syslog messages, but the functions and procedures provided by this module make it possible to solve more complex tasks which `pm_transformer` is not capable of on its own.

Structured data in IETF Syslog messages is parsed and put into NXLog fields. The SD-ID will be prepended to the field name with a dot unless it is `NXLOG@XXXX`. Consider the following Syslog message (line break added):

```
<30>1 2011-12-04T21:16:10.000000+02:00 host app procid msgid \
[NXLOG@32473 eventSource="Application" eventID="1011"] Message part
```

After this IETF-formatted Syslog message is parsed with `parse_syslog_ietf()`, there will be two additional fields: `$exampleSDID.eventID` and `$exampleSDID.eventSource`. When SD-ID is `NXLOG`, the field name will be the same as the SD-PARAM name. The two additional fields extracted from the structured data part of the following IETF Syslog message are `$eventID` and `$eventSource`:

```
<30>1 2011-12-04T21:16:10.000000+02:00 host app procid msgid \
[NXLOG@32473 eventSource="Application" eventID="1011"] Message part
```

All fields in the structured data part are parsed as `strings`.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.28.1. Configuration

The `xm_syslog` module accepts the following directives in addition to the [common module directives](#).

IETFTimestampInGMT

This is an alias for the `UTCTimestamp` directive below.

ReplaceLineBreaks

This optional directive specifies a character with which to replace line breaks in the Syslog message when generating Syslog events with `to_syslog_bsd()`, `to_syslog_ietf()`, and `to_syslog_snare()`. The default is a space. To retain line breaks in Syslog messages, set this to `\n`.

SnareDelimiter

This optional directive takes a single character (see [below](#)) as argument. This character is used by the `to_syslog_snare()` procedure to separate fields. If this directive is not specified, the default escape character is the tab (`\t`). In latter versions of Snare 4 this has changed to the hash mark (#); this directive can be used to specify the alternative delimiter. Note that there is no delimiter after the last field.

SnareReplacement

This optional directive takes a single character (see [below](#)) as argument. This character is used by the [to_syslog_snare\(\)](#) procedure to replace occurrences of the **delimiter** character inside the **\$Message** field. If this directive is not specified, the default replacement character is the space.

UTCTimestamp

This optional boolean directive can be used to format the timestamps produced by [to_syslog_ietf\(\)](#) in UTC/GMT instead of local time. The default is FALSE: local time is used with a timezone indicator.

86.28.1.1. Specifying Quote, Escape, and Delimiter Characters

The [SnareDelimiter](#) and [SnareReplacement](#) directives can be specified in several ways.

Unquoted single character

Any printable character can be specified as an unquoted character, except for the backslash (\):

```
Delimiter ;
```

Control characters

The following non-printable characters can be specified with escape sequences:

\a

audible alert (bell)

\b

backspace

\t

horizontal tab

\n

newline

\v

vertical tab

\f

formfeed

\r

carriage return

For example, to use TAB delimiting:

```
Delimiter \t
```

A character in single quotes

The configuration parser strips whitespace, so it is not possible to define a space as the delimiter unless it is enclosed within quotes:

```
Delimiter ' '
```

Printable characters can also be enclosed:

```
Delimiter ' ;'
```

The backslash can be specified when enclosed within quotes:

```
Delimiter '\'
```

A character in double quotes

Double quotes can be used like single quotes:

```
Delimiter " "
```

The backslash can be specified when enclosed within double quotes:

```
Delimiter "\\"
```

A hexadecimal ASCII code

Hexadecimal ASCII character codes can also be used by prepending `0x`. For example, the space can be specified as:

```
Delimiter 0x20
```

This is equivalent to:

```
Delimiter " "
```

86.28.2. Functions

The following functions are exported by `xm_syslog`.

`string syslog_facility_string(integer arg)`

Convert a Syslog facility value to a string.

`integer syslog_facility_value(string arg)`

Convert a Syslog facility string to an integer.

`string syslog_severity_string(integer arg)`

Convert a Syslog severity value to a string.

`integer syslog_severity_value(string arg)`

Convert a Syslog severity string to an integer.

86.28.3. Procedures

The following procedures are exported by `xm_syslog`.

`parse_syslog();`

Parse the `$raw_event` field as either BSD Syslog (RFC 3164) or IETF Syslog (RFC 5424) format.

`parse_syslog(string source);`

Parse the given string as either BSD Syslog (RFC 3164) or IETF Syslog (RFC 5424) format.

`parse_syslog_bsd();`

Parse the `$raw_event` field as BSD Syslog (RFC 3164) format.

`parse_syslog_bsd(string source);`

Parse the given string as BSD Syslog (RFC 3164) format.

`parse_syslog_ietf();`

Parse the `$raw_event` field as IETF Syslog (RFC 5424) format.

```
parse_syslog_ietf(string source);
```

Parse the given string as IETF Syslog (RFC 5424) format.

```
to_syslog_bsd();
```

Create a BSD Syslog formatted log message in `$raw_event` from the fields of the event. The following fields are used to construct the `$raw_event` field: `$EventTime`; `$Hostname`; `$SourceName`; `$ProcessID`; `$Message` or `$raw_event`; `$SyslogSeverity`, `$SyslogSeverityValue`, `$Severity`, or `$SeverityValue`; and `$SyslogFacility` or `$SyslogFacilityValue`. If the fields are not present, a sensible default is used.

```
to_syslog_ietf();
```

Create an IETF Syslog (RFC 5424) formatted log message in `$raw_event` from the fields of the event. The following fields are used to construct the `$raw_event` field: `$EventTime`; `$Hostname`; `$SourceName`; `$ProcessID`; `$Message` or `$raw_event`; `$SyslogSeverity`, `$SyslogSeverityValue`, `$Severity`, or `$SeverityValue`; and `$SyslogFacility` or `$SyslogFacilityValue`. If the fields are not present, a sensible default is used.

```
to_syslog_snare();
```

Create a SNARE Syslog formatted log message in `$raw_event`. The following fields are used to construct the `$raw_event` field: `$EventTime`, `$Hostname`, `$SeverityValue`, `$FileName`, `$EventID`, `$SourceName`, `$AccountName`, `$AccountType`, `$EventType`, `$Category` and `$Message`.

86.28.4. Fields

The following fields are used by `xm_syslog`.

In addition to the fields listed below, the `parse_syslog()` and `parse_syslog_ietf()` procedures will create fields from the Structured Data part of an IETF Syslog message. If the SD-ID in this case is not "NXLOG", these fields will be prefixed by the SD-ID (for example, `$mySDID.CustomField`).

\$raw_event (type: string)

A Syslog formatted string, set after `to_syslog_bsd()` or `to_syslog_ietf()` is called.

\$EventTime (type: datetime)

The timestamp found in the Syslog message, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called. If the year value is missing, it is set to the current year.

\$Hostname (type: string)

The hostname part of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called.

\$Message (type: string)

The message part of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called.

\$MessageID (type: string)

The MSGID part of the syslog message, set after `parse_syslog_ietf()` is called.

\$ProcessID (type: string)

The process ID in the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called.

\$Severity (type: string)

The normalized severity name of the event. See `$SeverityValue`.

\$SeverityValue (type: integer)

The normalized severity number of the event, mapped as follows.

Syslog Severity	Normalized Severity
0/emerg	5/critical
1/alert	5/critical
2/crit	5/critical
3/err	4/error
4/warning	3/warning
5/notice	2/info
6/info	2/info
7/debug	1/debug

\$SourceName (type: *string*)

The application/program part of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called.

\$SyslogFacility (type: *string*)

The facility name of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called. The default facility is `user`.

\$SyslogFacilityValue (type: *integer*)

The facility code of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called. The default facility is `1` (user).

\$SyslogSeverity (type: *string*)

The severity name of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called. The default severity is `notice`. See **\$SeverityValue**.

\$SyslogSeverityValue (type: *integer*)

The severity code of the Syslog line, set after `parse_syslog()`, `parse_syslog_bsd()`, or `parse_syslog_ietf()` is called. The default severity is `5` (notice). See **\$SeverityValue**.

86.28.5. Examples

Example 426. Sending a File as BSD Syslog over UDP

In this example, logs are collected from files, converted to BSD Syslog format with the `to_syslog_bsd()` procedure, and sent over UDP with the `om_udp` module.

```
nxlog.conf
1 <Extension syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input file>
6   Module im_file
7
8   # We monitor all files matching the wildcard.
9   # Every line is read into the $raw_event field.
10  File    "/var/log/app*.log"
11
12 <Exec>
13   # Set the $EventTime field usually found in the logs by
```

```
14      # extracting it with a regexp. If this is not set, the current
15      # system time will be used which might be a little off.
16      if $raw_event =~ /(\d\d\d\d\-\d\d-\d\d \d\d:\d\d:\d\d)/
17      {
18          $EventTime = parsedate($1);
19      }
20
21      # Now set the severity to something custom. This defaults to
22      # 'INFO' if unset.
23      if $raw_event =~ /ERROR/ $Severity = 'ERROR';
24      else $Severity = 'INFO';
25
26      # The facility can be also set, otherwise the default value is
27      # 'USER'.
28      $SyslogFacility = 'AUDIT';
29
30      # The SourceName field is called the TAG in RFC 3164
31      # terminology and is usually the process name.
32      $SourceName = 'my_application';
33
34      # It is also possible to rewrite the Hostname if you do not
35      # want to use the system hostname.
36      $Hostname = 'myhost';
37
38      # The Message field is used if present, otherwise the current
39      # $raw_event is prepended with the Syslog headers. You can do
40      # some modifications on the Message if required. Here we add
41      # the full path of the source file to the end of message line.
42      $Message = $raw_event + '[' + file_name() + ']';
43
44      # Now create our RFC 3164 compliant Syslog line using the
45      # fields set above and/or use sensible defaults where
46      # possible. The result will be in $raw_event.
47      to_syslog_bsd();
48  </Exec>
49 </Input>
50
51 <Output udp>
52      # This module just sends the contents of the $raw_event field to
53      # the destination defined here, one UDP packet per message.
54      Module om_udp
55      Host    192.168.1.42
56      Port    1514
57 </Output>
58
59 <Route file_to_udp>
60     Path    file => udp
61 </Route>
```

Example 427. Collecting BSD Style Syslog Messages over UDP

To collect BSD Syslog messages over UDP, use the [parse_syslog_bsd\(\)](#) procedure coupled with the [im_udp](#) module as in the following example.

nxlog.conf

```
1 <Extension syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input udp>
6   Module im_udp
7   Host 0.0.0.0
8   Port 514
9   Exec  parse_syslog_bsd();
10 </Input>
11
12 <Output file>
13   Module om_file
14   File  "/var/log/logmsg.txt"
15 </Output>
16
17 <Route syslog_to_file>
18   Path  udp => file
19 </Route>
```

Example 428. Collecting IETF Style Syslog Messages over UDP

To collect IETF Syslog messages over UDP as defined by RFC 5424 and RFC 5426, use the [parse_syslog_ietf\(\)](#) procedure coupled with the [im_udp](#) module as in the following example. Note that, as for BSD Syslog, the default port is 514 (as defined by RFC 5426).

nxlog.conf

```
1 <Extension syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input ietf>
6   Module im_udp
7   Host 0.0.0.0
8   Port 514
9   Exec  parse_syslog_ietf();
10 </Input>
11
12 <Output file>
13   Module om_file
14   File  "/var/log/logmsg.txt"
15 </Output>
16
17 <Route ietf_to_file>
18   Path  ietf => file
19 </Route>
```

Example 429. Collecting Both IETF and BSD Syslog Messages over the Same UDP Port

To collect both IETF and BSD Syslog messages over UDP, use the [parse_syslog\(\)](#) procedure coupled with the [im_udp](#) module as in the following example. This procedure is capable of detecting and parsing both Syslog formats. Since 514 is the default UDP port number for both BSD and IETF Syslog, this port can be useful to collect both formats simultaneously. To accept both formats on different ports, the appropriate parsers can be used as in the previous two examples.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input udp>
6     Module im_udp
7     Host   0.0.0.0
8     Port   514
9     Exec   parse_syslog();
10 </Input>
11
12 <Output file>
13     Module om_file
14     File    "/var/log/logmsg.txt"
15 </Output>
16
17 <Route syslog_to_file>
18     Path   udp => file
19 </Route>
```

Example 430. Collecting IETF Syslog Messages over TLS/SSL

To collect IETF Syslog messages over TLS/SSL as defined by RFC 5424 and RFC 5425, use the [parse_syslog_ietf\(\)](#) procedure coupled with the [im_ssl](#) module as in this example. Note that the default port is 6514 in this case (as defined by RFC 5425). The payload format parser is handled by the [Syslog_TLS](#) input reader.

nxlog.conf

```
1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input ssl>
6   Module      im_ssl
7   Host        localhost
8   Port        6514
9   CAFile     %CERTDIR%/ca.pem
10  CertFile    %CERTDIR%/client-cert.pem
11  CertKeyFile %CERTDIR%/client-key.pem
12  KeyPass     secret
13  InputType   Syslog_TLS
14  Exec        parse_syslog_ietf();
15 </Input>
16
17 <Output file>
18   Module      om_file
19   File        "/var/log/logmsg.txt"
20 </Output>
21
22 <Route ssl_to_file>
23   Path        ssl => file
24 </Route>
```

Example 431. Forwarding IETF Syslog over TCP

The following configuration uses the `to_syslog_ietf()` procedure to convert input to IETF Syslog and forward it over TCP.

nxlog.conf

```
1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input file>
6   Module      im_file
7   File        "/var/log/input.txt"
8   Exec        $TestField = "test value"; $Message = $raw_event;
9 </Input>
10
11 <Output tcp>
12   Module      om_tcp
13   Host        127.0.0.1
14   Port        1514
15   Exec        to_syslog_ietf();
16   OutputType  Syslog_TLS
17 </Output>
18
19 <Route file_to_syslog>
20   Path        file => tcp
21 </Route>
```

Because of the `Syslog_TLS` framing, the raw data sent over TCP will look like the following.

Output Sample

```
130 <13>1 2012-01-01T16:15:52.873750Z - - - [NXLOG@14506 EventReceivedTime="2012-01-01
17:15:52" TestField="test value"] test message<!
```

This example shows that all fields—except those which are filled by the Syslog parser—are added to the structured data part.

Example 432. Conditional Rewrite of the Syslog Facility—Version 1

If the message part of the Syslog event matches the regular expression, the `$SeverityValue` field will be set to the "error" Syslog severity integer value (which is provided by the `syslog_severity_value()` function).

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input udp>
6     Module im_udp
7     Port    514
8     Host    0.0.0.0
9     Exec    parse_syslog_bsd();
10 </Input>
11
12 <Output file>
13     Module om_file
14     File   "/var/log/logmsg.txt"
15     Exec   if $Message =~ /error/ $SeverityValue = syslog_severity_value("error");
16     Exec   to_syslog_bsd();
17 </Output>
18
19 <Route syslog_to_file>
20     Path   udp => file
21 </Route>
```

Example 433. Conditional Rewrite of the Syslog Facility—Version 2

The following example does almost the same thing as the previous example, except that the Syslog parsing and rewrite is moved to a processor module and the rewrite only occurs if the facility was modified. This can make processing faster on multi-core systems because the processor module runs in a separate thread. This method can also minimize UDP packet loss because the input module does not need to parse Syslog messages and therefore can process UDP packets faster.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input udp>
6     Module im_udp
7     Host   0.0.0.0
8     Port   514
9 </Input>
10
11 <Processor rewrite>
12     Module pm_null
13     <Exec>
14         parse_syslog_bsd();
15         if $Message =~ /error/
16         {
17             $SeverityValue = syslog_severity_value("error");
18             to_syslog_bsd();
19         }
20     </Exec>
21 </Processor>
22
23 <Output file>
24     Module om_file
25     File    "/var/log/logmsg.txt"
26 </Output>
27
28 <Route syslog_to_file>
29     Path   udp => rewrite => file
30 </Route>
```

86.29. W3C (xm_w3c)

This module provides a parser that can process data in the [W3C Extended Log File Format](#). It also understands the [BRO format](#) and Microsoft Exchange Message Tracking logs. While the [xm_csv](#) module can be used to parse these formats, xm_w3c has the advantage of automatically extracting information from the headers. This makes it much easier to parse such log files without the need to explicitly define the fields that appear in the input.

A common W3C log source is Microsoft IIS, which produces output like the following:

```
#Software: Microsoft Internet Information Services 7.0
#Version: 1.0
#Date: 2010-02-13 07:08:22
#Fields: date time s-sitename s-computername s-ip cs-method cs-uri-stem cs-uri-query s-port cs-
username c-ip cs-version cs(User-Agent) cs(Cookie) cs(Referer) cs-host sc-status sc-substatus sc-
win32-status sc-bytes cs-bytes time-taken
2010-02-13 07:08:21 W3SVC76 DNP1WEB1 174.120.30.2 GET / - 80 - 61.135.169.37 HTTP/1.1
Mozilla/5.0+(Windows;+U;+Windows+NT+5.1;+zh-CN;+rv:1.9.0.1)+Gecko/2008070208+Firefox/3.0.1 -
http://www.baidu.com/s?wd=QQ www.domain.com 200 0 0 29554 273 1452
2010-02-13 07:25:00 W3SVC76 DNP1WEB1 174.120.30.2 GET /index.htm - 80 - 119.63.198.110 HTTP/1.1
Baiduspider+(+http://www.baidu.jp/spider/) - - www.itcsoftware.com 200 0 0 17791 210 551
```

The format generated by BRO is similar, as it too defines the field names in the header. The field types and separator characters are also specified in the header. This allows the parser to automatically process the data. Below is a sample from BRO:

```
#separator \x09
#set_separator ,
#empty_field (empty)
#unset_field -
#path dns
#open 2013-04-09-21-01-43
#fields ts      uid      id.orig_h      id.orig_p      id.resp_h      id.resp_p      proto
trans_id    query    qclass   qclass_name    qtype   qtype_name    rcode   rcode_name   AA
TC          RD
RA          Z       answers TTLs
#types time      string    addr      port      addr      port      enum      count      string      count      string
count      string    count      string    bool      bool      bool      bool      count      vector[string]
vector[interval]
1210953058.350065      m2EJRWK7sCg      192.168.2.16      1920      192.168.2.1      53      udp
16995     ipv6.google.com 1      C_INTERNET      28      AAAA      0      NOERROR F      F      T
T          0
      ipv6.l.google.com,2001:4860:0:2001::68      8655.000000,300.000000
1210953058.350065      m2EJRWK7sCg      192.168.2.16      1920      192.168.2.1      53      udp
16995     ipv6.google.com 1      C_INTERNET      28      AAAA      0      NOERROR F      F      T
T          0
      ipv6.l.google.com,2001:4860:0:2001::68      8655.000000,300.000000
```

To use the parser in an input module, the [InputType](#) directive must reference the instance name of the xm_w3c module. See the [example below](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.29.1. Configuration

The xm_w3c module accepts the following directives in addition to the [common module directives](#).

Delimiter

This optional directive takes a single character (see [below](#)) as argument to specify the delimiter character used to separate fields. If this directive is not specified, the default delimiter character is either the space or tab character, as detected. For Microsoft Exchange Message Tracking logs the comma must be set as the delimiter:

```
Delimiter ,
```

Note that there is no delimiter after the last field in W3C, but Microsoft Exchange Message Tracking logs can contain a trailing comma.

FieldType

This optional directive can be used to specify a field type for a particular field. For example, to parse a **ByteSent** field as an integer, use **FieldType ByteSent integer**. This directive can be used more than once to provide types for multiple fields.

86.29.1.1. Specifying Quote, Escape, and Delimiter Characters

The **Delimiter** directive can be specified in several ways.

Unquoted single character

Any printable character can be specified as an unquoted character, except for the backslash (\):

```
Delimiter ;
```

Control characters

The following non-printable characters can be specified with escape sequences:

\a

audible alert (bell)

\b

backspace

\t

horizontal tab

\n

newline

\v

vertical tab

\f

formfeed

\r

carriage return

For example, to use TAB delimiting:

```
Delimiter \t
```

A character in single quotes

The configuration parser strips whitespace, so it is not possible to define a space as the delimiter unless it is enclosed within quotes:

```
Delimiter ' '
```

Printable characters can also be enclosed:

```
Delimiter ';'
```

The backslash can be specified when enclosed within quotes:

```
Delimiter '\\'
```

A character in double quotes

Double quotes can be used like single quotes:

```
Delimiter " "
```

The backslash can be specified when enclosed within double quotes:

```
Delimiter "\\"
```

A hexadecimal ASCII code

Hexadecimal ASCII character codes can also be used by prepending **0x**. For example, the space can be specified as:

```
Delimiter 0x20
```

This is equivalent to:

```
Delimiter " "
```

86.29.2. Fields

The following fields are used by xm_w3c.

\$EventTime (type: *datetime*)

Constructed from the *date* and *time* fields in the input, or from a *date-time* field.

\$SourceName (type: *string*)

The string in the *Software* header, such as **Microsoft Internet Information Services 7.0**.

86.29.3. Examples

Example 434. Parsing Advanced IIS Logs

The following configuration parses logs from the IIS Advanced Logging Module using the pipe delimiter. The logs are converted to JSON.

nxlog.conf

```
1 <Extension json>
2   Module      xm_json
3 </Extension>
4
5 <Extension w3cinput>
6   Module      xm_w3c
7   Delimiter   |
8 </Extension>
9
10 <Input w3c>
11   Module     im_file
12   File       'C:\inetpub\logs\LogFiles\W3SVC\ex*.log'
13   InputType  w3cinput
14 </Input>
15
16 <Output file>
17   Module     om_file
18   File       'C:\test\IIS.json'
19   Exec       to_json();
20 </Output>
21
22 <Route w3c_to_json>
23   Path       w3c => file
24 </Route>
```

86.30. WTMP (xm_wtmp)

This module provides a parser function to process binary wtmp files. The module registers a parser function using the name of the extension module instance. This parser can be used as a parameter for the **InputType** directive in input modules such as [im_file](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.30.1. Configuration

The *xm_wtmp* module accepts only the [common module directives](#).

86.30.2. Examples

Example 435. WTMP to JSON Format Conversion

The following configuration accepts WTMP and converts it to JSON.

nxlog.conf

```

1 <Extension wtmp>
2   Module      xm_wtmp
3 </Extension>
4
5 <Extension json>
6   Module      xm_json
7 </Extension>
8
9 <Input in>
10  Module     im_file
11    File      '/var/log/wtmp'
12    InputType wtmp
13    Exec      to_json();
14 </Input>
15
16 <Output out>
17   Module     om_file
18   File      '/var/log/wtmp.txt'
19 </Output>
20
21 <Route processwtmp>
22   Path      in => out
23 </Route>
```

Output Sample

```
{
  "EventTime": "2013-10-01 09:39:59",
  "AccountName": "root",
  "Device": "pts/1",
  "LoginType": "login",
  "EventReceivedTime": "2013-10-10 15:40:20",
  "SourceModuleName": "input",
  "SourceModuleType": "im_file"
}
{
  "EventTime": "2013-10-01 23:23:38",
  "AccountName": "shutdown",
  "Device": "no device",
  "LoginType": "shutdown",
  "EventReceivedTime": "2013-10-11 10:58:00",
  "SourceModuleName": "input",
  "SourceModuleType": "im_file"
}
```

86.31. XML (xm_xml)

This module provides functions and procedures for working with data formatted as Extensible Markup Language (XML). It can convert log messages to XML format and can parse XML into **fields**.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

86.31.1. Configuration

The `xm_xml` module accepts the following directives in addition to the [common module directives](#).

IgnoreRootTag

This optional boolean directive causes `parse_xml()` to omit the root tag when setting field names. For example, when this is set to TRUE and the `RootTag` is set to `Event`, a field might be named `$Event.timestamp`. With this directive set to FALSE, that field name would be `$timestamp`. The default value is TRUE.

ParseAttributes

When this optional boolean directive is set to TRUE, `parse_xml()` will also parse XML attributes. The default is FALSE (attributes are not parsed). For example, if `ParseAttributes` is set to TRUE, the following would be parsed into `$Msg.time`, `$Msg.type`, and `$Msg`:

```
<Msg time='2014-06-27T00:27:38' type='ERROR'>foo</Msg>
```

RootTag

This optional directive can be used to specify the name of the root tag that will be used by `to_xml()` to generate XML. The default `RootTag` is `Event`.

86.31.2. Functions

The following functions are exported by `xm_xml`.

`string to_xml()`

Convert the fields to XML and returns this as a string value. The `$raw_event` field and any field having a leading dot (.) or underscore (_) will be automatically excluded.

86.31.3. Procedures

The following procedures are exported by `xm_xml`.

`parse_xml();`

Parse the `$raw_event` field as XML input.

`parse_xml(string source);`

Parse the given string as XML format.

`to_xml();`

Convert the fields to XML and put this into the `$raw_event` field. The `$raw_event` field and any field having a leading dot (.) or underscore (_) will be automatically excluded.

86.31.4. Examples

Example 436. Syslog to XML Format Conversion

The following configuration accepts Syslog (both BSD and IETF) and converts it to XML.

nxlog.conf

```

1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension xml>
6     Module xm_xml
7 </Extension>
8
9 <Input tcp>
10    Module im_tcp
11    Port 1514
12    Host 0.0.0.0
13    Exec parse_syslog(); to_xml();
14 </Input>
15
16 <Output file>
17    Module om_file
18    File "/var/log/log.xml"
19 </Output>
20
21 <Route tcp_to_file>
22    Path tcp => file
23 </Route>
```

Input Sample

```
<30>Sep 30 15:45:43 host44.localdomain.hu acpid: 1 client rule loaded
```

Output Sample

```

<Event>
  <MessageSourceAddress>127.0.0.1</MessageSourceAddress>
  <EventReceivedTime>2012-03-08 15:05:39</EventReceivedTime>
  <SyslogFacilityValue>3</SyslogFacilityValue>
  <SyslogFacility>DAEMON</SyslogFacility>
  <SyslogSeverityValue>6</SyslogSeverityValue>
  <SyslogSeverity>INFO</SyslogSeverity>
  <SeverityValue>2</SeverityValue>
  <Severity>INFO</Severity>
  <Hostname>host44.localdomain.hu</Hostname>
  <EventTime>2012-09-30 15:45:43</EventTime>
  <SourceName>acpid</SourceName>
  <Message>1 client rule loaded</Message>
</Event>
```

Example 437. Converting Windows EventLog to Syslog-Encapsulated XML

The following configuration reads the Windows EventLog and converts it to the BSD Syslog format where the message part contains the fields in XML.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Extension xml>
6     Module xm_xml
7 </Extension>
8
9 <Input eventlog>
10    Module im_msvisalog
11    Exec   $Message = to_xml(); to_syslog_bsd();
12 </Input>
13
14 <Output tcp>
15    Module om_tcp
16    Host   192.168.1.1
17    Port   1514
18 </Output>
19
20 <Route eventlog_to_tcp>
21    Path   eventlog => tcp
22 </Route>
```

Output Sample

```
<14>Mar  8 15:12:12 WIN-OUNNPISDHIG Service_Control_Manager: <Event><EventTime>2012-03-08
15:12:12</EventTime><EventTimeWritten>2012-03-08 15:12:12</EventTimeWritten><Hostname>WIN-
OUNNPISDHIG</Hostname><EventType>INFO</EventType><SeverityValue>2</SeverityValue><Severity>INFO
</Severity><SourceName>Service Control
Manager</SourceName><FileName>System</FileName><EventID>7036</EventID><CategoryNumber>0</Catego
ryNumber><RecordNumber>6791</RecordNumber><Message>The nxlog service entered the running state.
</Message><EventReceivedTime>2012-03-08 15:12:14</EventReceivedTime></Event>↔
```

Chapter 87. Input Modules

Input modules are responsible for collecting event log data from various sources.

Each module provides a set of fields for each log message, these are documented in the corresponding sections below. The NXLog core will add to this set the fields listed in the following section.

87.1. Fields

The following fields are used by core.

`$raw_event` (type: *string*)

The data received from stream modules (im_file, im_tcp, etc.).

`$EventReceivedTime` (type: *datetime*)

The time when the event is received. The value is not modified if the field already exists.

`$SourceModuleName` (type: *string*)

The name of the module instance, for input modules. The value is not modified if the field already exists.

`$SourceModuleType` (type: *string*)

The type of module instance (such as `im_file`), for input modules. The value is not modified if the field already exists.

87.2. BSD/Linux Process Accounting (im_acct)

This module can be used to collect process accounting logs from a Linux or BSD kernel.

87.2.1. Configuration

The `im_acct` module accepts the following directives in addition to the [common module directives](#).

`AcctOff`

This boolean directive specifies that accounting should be disabled when `im_acct` stops. If **AcctOff** is set to FALSE, accounting will not be disabled; events will continue to be written to the log file for NXLog to collect later. The default is FALSE.

`AcctOn`

This boolean directive specifies that accounting should be enabled when `im_acct` starts. If **AcctOn** is set to FALSE, accounting will not be enabled automatically. The default is TRUE.

`File`

This optional directive specifies the path where the kernel writes accounting data.

`FileSizeLimit`

NXLog will automatically truncate the log file when it reaches this size, specified as an integer in bytes (see [Integer](#)). The default is 1 MB.

87.2.2. Fields

The following fields are used by im_acct.

`$raw_event` (type: *string*)

A string containing a list of key/value pairs from the event.

\$btime (type: *datetime*)

The process start time.

\$comm (type: *string*)

The first 16 characters of the command name.

\$compat (type: *boolean*)

Set to TRUE if a **COMPAT** flag is associated with the process event (used compatibility mode).

\$core (type: *boolean*)

Set to TRUE if a **CORE** flag is associated with the process event (dumped core).

\$etime (type: *string*)

The total elapsed time.

\$exitcode (type: *integer*)

The process exit code.

\$fork (type: *boolean*)

Set to TRUE if a **FORK** flag is associated with the process event (has executed fork, but no exec).

\$gid (type: *integer*)

The group ID of the process.

\$group (type: *string*)

The system group corresponding to the **\$gid**.

\$io (type: *string*)

The characters transferred.

\$majflt (type: *string*)

The number of major page faults.

\$mem (type: *integer*)

The average memory usage of the process (on BSD).

\$mem (type: *string*)

The average memory usage of the process (on Linux).

\$minflt (type: *string*)

The number of minor page faults.

\$rw (type: *string*)

The number of blocks read or written.

\$stime (type: *string*)

The total system processing time elapsed.

\$su (type: *boolean*)

Set to TRUE if a **SU** flag is associated with the process event (used superuser privileges).

\$uid (type: *integer*)

The user ID of the process.

\$user (type: *string*)

The system user corresponding to the **\$uid**.

\$utime (type: *string*)

The total user processing time elapsed.

\$xsig (type: *boolean*)

Set to TRUE if an **XSIG** flag is associated with the process event (killed by a signal).

87.2.3. Examples

Example 438. Collecting Process Accounting Logs

With this configuration, the *im_acct* module will collect process accounting logs. Process accounting will be automatically enabled and configured to write logs to the file specified. NXLog will allow the file to grow to a maximum size of 10 MB before truncating it.

```
nxlog.conf
1 <Input acct>
2   Module      im_acct
3   File        '/var/log/acct.log'
4   fileSizeLimit 10M
5 </Input>
```

87.3. AIX Auditing (*im_aixaudit*)

This module parses events in the AIX Audit format. This module reads directly from the kernel. See also [xm_aixaudit](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.3.1. Configuration

The *im_aixaudit* module accepts the following directives in addition to the [common module directives](#).

DeviceFile

This optional directive specifies the device file from which to read audit events. If this is not specified, it defaults to **/dev/audit**.

87.3.2. Fields

See the [xm_aixaudit Fields](#).

87.3.3. Examples

Example 439. Reading AIX Audit Events From the Kernel

This configuration reads AIX audit events directly from the kernel via the (default) `/dev/audit` device file.

`nxlog.conf`

```
1 <Input in>
2     Module      im_aixaudit
3     DeviceFile  /dev/audit
4 </Input>
```

87.4. Azure (im_azure)

This module can be used to collect logs from [Microsoft Azure](#) applications.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.4.1. Storage Setup

Azure web application logging and storage can be configured with the [Azure Management Portal](#).

1. After logging in to the Portal, click **New** on the left panel, select the **Storage** category, and choose the **Storage account - blob, file, table, queue**.
2. Create the new storage account. Provide a storage name, location, and replication type.
3. Click **[Create Storage Account]** and wait for storage setup to complete.
4. Go to **Apps**, select the application for which to enable logging, and click **Configure**.
5. Scroll down to the application diagnostic section and configure the table and blob storage options corresponding with the storage account created above.
6. Confirm the changes by clicking **Save**, then restart the service. Note that it may take a while for Azure to create the table and/or blob in the storage.

87.4.2. Configuration

The `im_azure` module accepts the following directives in addition to the [common module directives](#). The `AuthKey` and `StorageName` directives are required, along with either `BlobName` or `TableName`.

`AuthKey`

This mandatory directive specifies the authentication key to use for connecting to Azure.

`BlobName`

This directive specifies the storage blob to connect to. One of `BlobName` and `TableName` must be defined (but not both).

`StorageName`

This mandatory directive specifies the name of the storage account from which to collect logs.

`TableName`

This directive specifies the storage table to connect to. One of `BlobName` and `TableName` must be defined (but not both).

`Address`

This directive specifies the URL for connecting to the storage account and corresponding table or blob. If this

directive is not specified, it defaults to `http://<table|blob>.<storagename>.core.windows.net`. If defined, the value must start with `http://` or `https://`.

HTTPSAallowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with an unknown or self-signed certificate. The default value is FALSE: all HTTPS connections must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote HTTPS client. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote HTTPS client.

HTTPSCertFile

This specifies the path of the certificate file to be used for the HTTPS handshake.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the HTTPS handshake.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote HTTPS client. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote HTTPS client.

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

HTTPSRequireCert

This boolean directive specifies that the remote HTTPS client must present a certificate. If set to TRUE and there is no certificate presented during the connection handshake, the connection will be refused. The default value is TRUE: each connection must use a certificate.

HTTPSSSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: `SSLv2`, `SSLv3`, `TLSv1`, `TLSv1.1`, and `TLSv1.2`. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
HTTPSSSLProtocol TLSv1.2
```

PollInterval

This directive specifies how frequently the module will check for new events, in seconds. If this directive is not

specified, it defaults to 1 second. Fractional seconds may be specified (`PollInterval 0.5` will check twice every second).

87.4.3. Fields

The following fields are used by im_azure.

`$raw_event` (type: `string`)

The raw string from the event.

`$EventTime` (type: `datetime`)

The timestamp of the event.

`$ProcessID` (type: `integer`)

The ID of the process which generated the event.

`$Severity` (type: `string`)

The severity of the event, if available. The severity is mapped as follows.

Azure Severity	Normalized Severity
Critical	5/CRITICAL
Warning	3/WARNING
Information	2/INFO
Verbose	1/DEBUG

`$SeverityValue` (type: `integer`)

The severity value of the event, if available; see the `$Severity` field.

`$SourceName` (type: `string`)

The name of the application which generated the event, if available.

`$ThreadID` (type: `integer`)

The ID of the thread which generated the event.

87.5. Batched Compression over TCP or SSL (im_batchcompress)

The `im_batchcompress` module provides a compressed network transport with optional SSL encryption. It uses its own protocol to receive and decompress a batch of messages sent by [om_batchcompress](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.5.1. Configuration

The `im_batchcompress` module accepts the following directives in addition to the [common module directives](#).

`ListenAddr`

The module will accept connections on this IP address or DNS hostname. The default is `localhost`.

`Port`

The module instance will listen on this port for incoming connections. The default is port 2514.

AllowUntrusted

This boolean directive specifies whether the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with an unknown or self-signed certificate. The default value is FALSE: by default, all connections must present a trusted certificate.

CADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format.

CAFfile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote socket.

CertFile

This specifies the path of the certificate file to be used for the SSL handshake.

CertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake.

CRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote socket. The filenames in this directory must be in the OpenSSL hashed format.

CRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote socket.

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [CertKeyFile](#). This directive is not needed for passwordless private keys.

RequireCert

This boolean directive specifies that the remote must present a certificate. If set to TRUE and there is no certificate presented during the connection handshake, the connection will be refused. The default value is TRUE: by default, each connections must use a certificate.

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) could use SSLv3 only and did not support TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

87.5.2. Fields

The following fields are used by im_batchcompress.

\$MessageSourceAddress (type: *string*)

The IP address of the remote host.

87.5.3. Examples

Example 440. Reading Batch Compressed Data

This configuration listens on port 2514 for incoming log batches and writes them to file.

nxlog.conf

```
1 <Input batchcompress>
2   Module      im_batchcompress
3   ListenAddr  0.0.0.0
4   Port        2514
5 </Input>
6
7 <Output file>
8   Module      om_file
9   File        "tmp/output"
10 </Output>
11
12 <Route batchcompress_to_file>
13   Path        batchcompress => file
14 </Route>
```

87.6. Basic Security Module Auditing (im_bsm)

This module provides support for parsing events logged using Sun's Basic Security Module (BSM) Auditing API. This module reads directly from the kernel. See also [xm_bsm](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.6.1. Configuration

The *im_bsm* module accepts the following directives in addition to the [common module directives](#).

DeviceFile

This optional directive specifies the device file from which to read BSM events. If this is not specified, it defaults to [/dev/auditpipe](#).

87.6.2. Fields

See the *xm_bsm* [Fields](#).

87.6.3. Examples

Example 441. Reading BSM Audit Events From the Kernel

This configuration reads BSM audit events directly from the kernel via the (default) `/dev/auditpipe` device file.

nxlog.conf

```
1 <Input in>
2   Module      im_bsm
3   DeviceFile  /dev/auditpipe
4 </Input>
```

87.7. CheckPoint OPSEC LEA (im_checkpoint)

This module provides support for collecting logs remotely from CheckPoint devices over the [OPSEC LEA](#) protocol. The OPSEC LEA protocol makes it possible to establish a trusted secure and authenticated connection with the remote device.

NOTE

The OPSEC SDK provides libraries only in 32-bit versions and this makes it impossible to compile a 64-bit application. For this reason the *im_checkpoint* module uses a helper program called *nx-im-checkpoint*. This helper is responsible for collecting the logs and transmitting these over a pipe to the *im_checkpoint* module.

CheckPoint uses a certificate export method with an activation password so that certificate keys can be securely transferred over the network in order to establish trust relationships between the entities involved when using SSL-based authenticated connections. The following entities (hosts) are involved in the log generation and collection process:

SmartDashboard

The firewall administrator can use the SmartDashboard management interface to connect to and manage the firewall.

SecurePlatform based FireWall-1

The SecurePlatform based FireWall-1 device will be generating the logs (SPLAT).

NXLog

The log collector running NXLog which connects to SPLAT over the OPSEC LEA protocol utilizing the *im_checkpoint* module.

The following steps are required to configure the LEA connection between SPLAT and NXLog.

1. Enable the LEA service on SPLAT. Log in to SPLAT, enter `expert` mode, and run `vi $FWDIR/conf/fwopsec.conf`. Make sure the file contains the following lines. Then restart the firewall with the `cprestart` command (or `cpstop` and `cpstart`).

fwopsec.conf

```
lea_server auth_port 18184
lea_server auth_type sslca
```

2. Make sure SPLAT will accept ICA pull requests, the LEA Connection (port 18184), and can generate logs. For

testing purposes, it is easiest to create a single rule to accept all connections and log these. For this the SmartDashboard host must be added as a GUI Client on SPLAT and a user needs to be configured to be able to log onto SPLAT remotely from SmartDashboard.

3. Create the certificates for NXLog in SmartDashboard. Select **Manage > Servers > OPSEC Applications**, then click **[New]** and select **OPSEC Application**. A dialog window should appear. Fill in the following properties and then click **[OK]**.

Name

Set to **nxlog**.

Description

Set to **NXLog log collector** or something similar.

Host

Click on **[New]** to create a new host and name it accordingly (**nxloghost**, for example).

Client Entities

Check **LEA**. All other options should be unchecked.

Secure Internal Communication

Click on **[Communication]**. Another dialog window will appear. Enter and re-enter the activation keys, then click **[Initialize]**. Trust state should change from *Uninitialized* to *Initialized but trust not established*. Click **[Close]**. Now in the OPSEC Application Properties window the *DN* should appear. This generated string looks like this: **CN=nxlog,0=splat..ebo9pf**. This value will be used in our **lea.conf** file as the **opsec_sic_name** parameter.

4. Retrieve the OPSEC application certificate. From the NXLog host, run the following command: **/opt/nxlog/bin/opsec_pull_cert -h SPLAT_IP_ADDR -n nxlog -p ACTIVATION_KEY**. Make sure to substitute the correct values in place of **SPLAT_IP_ADDR** and **ACTIVATION_KEY**. If the command is successful, the certificate file **opsec.p12** should appear in the current directory. Copy this file to **/opt/nxlog/etc**.
5. Get the *DN* of SPLAT. In SmartDashboard, double-click on **Network Objects > Check Point > SPLAT**. The properties window will contain a similar *DN* under *Secure Internal Communication* such as **CN=cp_mgmt,o=splat..ebo9pf**.
6. Retrieve the **sic_policy.conf** file from SPLAT. Initiate a secure copy from the firewall in expert mode. Then move the file to the correct location.

```
[Expert@checkpoint]# scp $CPDIR/conf/sic_policy.conf user@rhel:/home/user  
[root@rhel ~]# mv /home/user/sic_policy.conf /opt/nxlog/etc
```

7. Edit **/opt/nxlog/etc/sic_policy.conf**, and add the necessary policy to the **[Outbound rules]** section.

sic_policy.conf

```
1 [Outbound rules]  
2 # apply_to peer(s) port(s) service(s) auth-method(s)  
3 # -----  
4  
5 # OPSEC configurations - place here (and in [Inbound rules] too)  
6 ANY ; ANY ; 18184 ; fwn1_opsec, ssl_opsec, ssl_clear_opsec, lea ; any_method
```

8. Edit **/opt/nxlog/etc/lea.conf**. The file should contain the following. Make sure to substitute the correct value in place of **SPLAT_IP_ADDR** and use the correct DN values for **opsec_sic_name** and **lea_server opsec_entity_sic_name**.

lea.conf

```
lea_server ip SPLAT_IP_ADDR
lea_server auth_port 18184
lea_server auth_type sslca
opsec_sic_name "CN=nxlog,O=splat..ebo9pf"
opsec_sslca_file /opt/nxlog/etc/opsec.p12
lea_server opsec_entity_sic_name "CN=cp_mgmt,O=splat..ebo9pf"
opsec_sic_policy_file /opt/nxlog/etc/sic_policy.conf
```

Refer to the CheckPoint documentation for more information regarding the LEA log service configuration.

To test whether the log collection works, execute the following command: `/opt/nxlog/bin/nx-im-checkpoint --readfromlast FALSE > output.bin`. The process should not exit. Type **Ctrl+c** to interrupt it. The created file `output.bin` should contain logs in NXLog's **Binary** format.

NOTE

The `OPSEC_DEBUG_LEVEL` environment variable can be set to get debugging information if something goes wrong and there is no output produced. Run `OPSEC_DEBUG_LEVEL=1 /opt/nxlog/bin/nx-im-checkpoint --readfromlast FALSE > output.bin` and check the debug logs printed to standard error.

NOTE

The two files `sslauthkeys.C` and `sslsess.C` are used during the key-based authentication. These files are stored in the same directory where `lea.conf` resides. To override this, set the `OPSECDIR` environment variable.

If the log collection is successful, you can now try running NXLog with the *im_checkpoint* module.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.7.1. Configuration

The *im_checkpoint* module accepts the following directives in addition to the [common module directives](#).

Command

The optional directive specifies the path of the `nx-im-checkpoint` binary. If not specified, the default is `/opt/nxlog/bin/nx-im-checkpoint` on Linux.

LEAConfigFile

This optional directive specifies the path of the LEA configuration file. If not specified, the default is `/opt/nxlog/etc/lea.conf`. This file must be edited in order for the OPSEC LEA connection to work.

LogFile

This can be used to specify the log file to be read. If not specified, it defaults to `fw.log`. To collect the audit log, use `LogFile fw.adtlog` which would then be passed to the `nx-im-checkpoint` binary as `--logfile fw.adtlog`.

ReadFromLast

This optional boolean directive instructs the module to only read logs which arrived after NXLog was started if the saved position could not be read (for example on first start). When `SavePos` is TRUE and a previously saved record number could be read, the module will resume reading from this saved record number. If **ReadFromLast** is FALSE, the module will read all logs from the LEA source. This can result in quite a lot of messages, and is usually not the expected behavior. If this directive is not specified, it defaults to TRUE.

Restart

Restart the **nx-im-checkpoint** process if it exits. There is a one second delay before it is restarted to avoid a denial-of-service if the process is not behaving. This boolean directive defaults to FALSE.

SavePos

This boolean directive specifies that the last record number should be saved when NXLog exits. The record number will be read from the cache file upon startup. The default is TRUE: the record number is saved if this directive is not specified. Even if SavePos is enabled, it can be explicitly turned off with the global **NoCache** directive.

87.7.2. Fields

The following fields are used by im_checkpoint.

The LEA protocol provides CheckPoint device logs in a structured format. For the list of LEA fields, see [LEA Fields Update](#) on CheckPoint.com. Some of the field names are mapped to normalized names which NXLog uses in other modules (such as **\$EventTime**). The list of these fields is provided below. The other LEA fields are reformatted such that non-alphanumeric characters are replaced with an underscore (**_**) in field names. The **\$raw_event** field contains the list of all fields and their respective values without any modification to the original LEA field naming.

\$raw_event (type: *string*)

Contains the **\$EventTime**, **\$Hostname**, **\$Severity**, and the original LEA fields in **fieldname=value** pairs.

\$AccountName (type: *string*)

The user name. Originally called *user*.

\$ApplicationName (type: *string*)

The application that the user is trying to access. Originally called *app_name*.

\$DestinationIPv4Address (type: *ip4addr*)

The destination IP address of the connection. Originally called *dst*.

\$DestinationPort (type: *integer*)

The destination port number. Originally called *d_port*.

\$Direction (type: *string*)

The direction of the connection with respect to the interface. Can be either **inbound** or **outbound**. Originally called *i/f_dir*.

\$EventDuration (type: *string*)

The duration of the connection. Originally called *elapsed*.

\$EventTime (type: *datetime*)

The date and time of the event. Originally called *time*.

\$Hostname (type: *string*)

The IP address or hostname of the device which generated the log. Originally called *orig*.

\$Interface (type: *string*)

The name of the interface the connection passed through. Originally called *i/f_name*.

\$RecordNumber (type: *integer*)

The record number which identifies the log entry. Originally called *loc*.

\$Severity (type: *string*)

The IPS protection severity level setting. Originally called *severity*. Set to **INFO** if it was not provided in the logs.

\$SourceIPv4Address (type: *ip4addr*)

The source IP address of the connection. Originally called *src*.

\$SourceName (type: *string*)

The name of the device which generated the log. Originally called *product*.

\$SourcePort (type: *integer*)

The source port number of the connection. Originally called *s_port*.

87.7.3. Examples

Example 442. Converting CheckPoint LEA Input to JSON

This configuration instructs NXLog to collect logs from CheckPoint devices over the LEA protocol and store the logs in a file in JSON format.

nxlog.conf

```

1 <Extension json>
2   Module      xm_json
3 </Extension>
4
5 <Input checkpoint>
6   Module      im_checkpoint
7   Command     /opt/nxlog/bin/nx-im-checkpoint
8   LEAConfigFile /opt/nxlog/etc/lea.conf
9 </Input>
10
11 <Output file>
12   Module      om_file
13   File        'tmp/output'
14   Exec        $raw_event = to_json();
15 </Output>
16
17 <Route checkpoint_to_file>
18   Path        checkpoint => file
19 </Route>
```

87.8. DBI (im_dbi)

The *im_dbi* module allows NXLog to pull log data from external databases. This module utilizes the [libdbi](#) database abstraction library, which supports various database engines such as MySQL, PostgreSQL, MSSQL, Sybase, Oracle, SQLite, and Firebird. A SELECT statement can be specified, which will be executed periodically to check for new records.

NOTE

The *im_dbi* and [*om_dbi*](#) modules support GNU/Linux only because of the libdbi library. The [*im_odbc*](#) and [*om_odbc*](#) modules provide native database access on Windows.

NOTE

libdbi needs [drivers](#) to access the database engines. These are in the libdbd-* packages on Debian and Ubuntu. CentOS 5.6 has a libdbi-drivers RPM package, but this package does not contain any driver binaries under /usr/lib64/dbd. The drivers for both MySQL and PostgreSQL are in libdbi-db-mysql. If these are not installed, NXLog will return a libdbi driver initialization error.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.8.1. Configuration

The *im_db* module accepts the following directives in addition to the [common module directives](#).

Driver

This mandatory directive specifies the name of the libdbi driver which will be used to connect to the database. A DRIVER name must be provided here for which a loadable driver module exists under the name **libdbdDRIVER.so** (usually under `/usr/lib/dbd/`). The MySQL driver is in the **libdbdmysql.so** file.

SQL

This directive should specify the SELECT statement to be executed every **PollInterval** seconds. The module automatically appends a **WHERE id > ? LIMIT 10** clause to the statement. The result set returned by the SELECT statement must contain an *id* column which is then stored and used for the next query.

Option

This directive can be used to specify additional driver options such as connection parameters. The manual of the libdbi driver should contain the options available for use here.

PollInterval

This directive specifies how frequently the module will check for new records, in seconds. If this directive is not specified, the default is 1 second. Fractional seconds may be specified (**PollInterval 0.5** will check twice every second).

SavePos

If this boolean directive is set to TRUE, the position will be saved when NXLog exits. The position will be read from the cache file upon startup. The default is TRUE: the position will be saved if this directive is not specified. Even if **SavePos** is enabled, it can be explicitly turned off with the global **NoCache** directive.

87.8.2. Examples

Example 443. Reading From a MySQL Database

This example uses libdbi and the MySQL driver to connect to the logdb database on the local host and execute the provided statement.

nxlog.conf

```
1 <Input dbi>
2   Module im_db
3   Driver mysql
4   Option host 127.0.0.1
5   Option username mysql
6   Option password mysql
7   Option dbname logdb
8   SQL    SELECT id, facility, severity, hostname, \
9           timestamp, application, message \
10          FROM log
11 </Input>
12
13 <Output file>
14   Module om_file
15   File   "tmp/output"
16 </Output>
17
18 <Route dbi_to_file>
19   Path   dbi => file
20 </Route>
```

87.9. Event Tracing for Windows (im_etw)

This module can be used to collect events through [Event Tracing for Windows \(ETW\)](#).

ETW is a mechanism in Windows designed for efficient logging of both kernel and user-mode applications. Debug and Analytical channels are based on ETW and cannot be collected as regular Windows Eventlog channels via the [im_msvisalog](#) module. Various Windows services such as the Windows Firewall and DNS Server can be configured to log events through Windows Event Tracing.

The *im_etw* module reads event tracing data directly for maximum efficiency. Unlike other solutions, *im_etw* does not save ETW data into intermediary trace files that need to be parsed again.

NOTE The *im_etw* module is only available on the Windows platform.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.9.1. Configuration

The *im_etw* module accepts the following directives in addition to the [common module directives](#). One of **KernelFlags** and **Provider** must be specified.

KernelFlags

This directive specifies that kernel trace logs should be collected, and accepts a comma-separated list of flags to use for filtering the logs. The **Provider** and **KernelFlags** directives are mutually exclusive (but one must be specified). The following values are allowed: **ALPC**, **CSWITCH**, **DBGPRINT**, **DISK_FILE_IO**, **DISK_IO**, **DISK_IO_INIT**, **DISPATCHER**, **DPC**, **DRIVER**, **FILE_IO**, **FILE_IO_INIT**, **IMAGE_LOAD**, **INTERRUPT**, **MEMORY_HARDFAULTS**, **MEMORY_PAGEFAULTS**, **NETWORK_TCPIP**, **NO_SYSCONFIG**, **PROCESS**, **PROCESS_COUNTERS**, **PROFILE**, **REGISTRY**, **SPLIT_IO**, **SYSTEMCALL**, **THREAD**, **VAMAP**, and **VIRTUAL_ALLOC**.

Provider

This directive specifies the name (not GUID) of the ETW provider from which to collect trace logs. Providers available for tracing can be listed with `logman query providers`. The **Provider** and **KernelFlags** directives are mutually exclusive (but one must be specified). The **Windows Kernel Trace** provider is not supported; instead, the **KernelFlags** directive should be used to open a kernel logger session.

Level

This optional directive specifies the log level for collecting trace events. Because kernel log sessions do not provide log levels, this directive is only available in combination with the **Provider** directive. Valid values include **Critical**, **Error**, **Warning**, **Information**, and **Verbose**. If this directive is not specified, the verbose log level is used.

87.9.2. Fields

The following fields are used by im_etw.

Any additional fields defined by the trace provider will also be included in the event record.

`$raw_event` (type: *string*)

A string containing a **field=value** pair for each field in the event.

`$AccountName` (type: *string*)

The username associated with the event.

`$AccountType` (type: *string*)

The type of the account. Possible values are: **User**, **Group**, **Domain**, **Alias**, **Well Known Group**, **Deleted Account**, **Invalid**, **Unknown**, and **Computer**.

`$ActivityID` (type: *string*)

The ID of the activity corresponding to the event.

`$Channel` (type: *integer*)

The channel to which the event log should be directed.

`$Domain` (type: *string*)

The domain name of the user.

`$EventId` (type: *integer*)

The Event ID, corresponding to the provider, that indicates the type of event.

`$EventTime` (type: *datetime*)

The time when the event was generated.

`$EventType` (type: *string*)

One of **CRITICAL**, **ERROR**, **WARNING**, **DEBUG**, **AUDIT_FAILURE**, **AUDIT_SUCCESS**, or **INFO**.

`$ExecutionProcessID` (type: *integer*)

The ID of the process that generated the event.

`$ExecutionThreadID` (type: *integer*)

The ID of the thread that generated the event.

\$Keywords (type: *string*)

A keyword bit mask corresponding to the current event.

\$OpcodeValue (type: *integer*)

An integer indicating the operation corresponding to the event.

\$ProviderGuid (type: *string*)

The GUID of the trace provider, corresponding to the [\\$SourceName](#).

\$Severity (type: *string*)

The normalized severity name of the event. See [\\$SeverityValue](#).

\$SeverityValue (type: *integer*)

The normalized severity number of the event, mapped as follows.

Event Log Severity	Normalized Severity
0/Audit Success	2/INFO
0/Audit Failure	4/ERROR
1/Critical	5/CRITICAL
2/Error	4/ERROR
3/Warning	3/WARNING
4/Information	2/INFO
5/Verbose	1/DEBUG

\$SourceName (type: *string*)

The name of the trace provider.

\$TaskValue (type: *integer*)

An integer indicating a particular component of the provider.

\$Version (type: *integer*)

The version of the event type.

87.9.3. Examples

Example 444. Collecting Events From the Windows Kernel Trace

With this configuration, NXLog will collect trace events from the Windows kernel. Only events matching the **PROCESS** and **THREAD** flags will be collected.

```
nxlog.conf
1 <Input etw>
2   Module      im_etw
3   KernelFlags PROCESS, THREAD
4 </Input>
```

Example 445. Collecting Events From an ETW Provider

With this configuration, NXLog will collect events from the **Windows-Windows-DNSServer** trace provider.

nxlog.conf

```
1 <Input etw>
2     Module      im_etw
3     Provider    Windows Kernel Trace
4 </Input>
```

87.10. Program (im_exec)

This module will execute a program or script on startup and read its standard output. It can be used to easily integrate with exotic log sources which can be read only with the help of an external script or program.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

WARNING

If you are using a Perl script, consider using [im_perl](#) instead or turning on *Autoflush* with `$|=1;`, otherwise *im_exec* might not receive data immediately due to Perl's internal buffering. See the [Perl language reference](#) for more information about `$|`.

87.10.1. Configuration

The *im_exec* module accepts the following directives in addition to the [common module directives](#). The [Command](#) directive is required.

Command

This mandatory directive specifies the name of the program or script to be executed.

Arg

This is an optional parameter. **Arg** can be specified multiple times, once for each argument that needs to be passed to the [Command](#). Note that specifying multiple arguments with one **Arg** directive, with arguments separated by spaces, will not work (the [Command](#) would receive it as one argument).

InputType

See the [InputType](#) description in the global module configuration section.

Restart

Restart the process if it exits. There is a one second delay before it is restarted to avoid a denial-of-service when a process is not behaving. Looping should be implemented in the script itself, this directive is only to provide some safety against malfunctioning scripts and programs. This boolean directive defaults to FALSE: the [Command](#) will not be restarted if it exits.

87.10.2. Examples

Example 446. Emulating *im_file*

This configuration uses the tail command to read from a file.

NOTE

The `im_file` module should be used to read log messages from files. This example only demonstrates the use of the `im_exec` module.

nxlog.conf

```
1 <Input messages>
2   Module im_exec
3   Command /usr/bin/tail
4   Arg    -f
5   Arg    /var/log/messages
6 </Input>
7
8 <Output file>
9   Module om_file
10  File   "tmp/output"
11 </Output>
12
13 <Route messages_to_file>
14   Path   messages => file
15 </Route>
```

87.11. File (*im_file*)

This module can be used to read log messages from files. The file position can be persistently saved across restarts in order to avoid reading from the beginning again when NXLog is restarted. External rotation tools are also supported. When the module is not able to read any more data from the file, it checks whether the opened file descriptor belongs to the same filename it opened originally. If the inodes differ, the module assumes the file was moved and reopens its input.

im_file uses a one second interval to monitor files for new messages. This method was implemented because polling a regular file is not supported on all platforms. If there is no more data to read, the module will sleep for 1 second.

By using wildcards, the module can read multiple files simultaneously and will open new files as they appear. It will also enter newly created directories if recursion is enabled.

NOTE

The module needs to scan the directory content for wildcarded file monitoring. This can present a significant load if there are many files (hundreds or thousands) in the monitored directory. For this reason it is highly recommended to rotate files out of the monitored directory either using the built-in log rotation capabilities of NXLog or with external tools.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.11.1. Configuration

The `im_file` module accepts the following directives in addition to the [common module directives](#). The `File` directive is required.

File

This mandatory directive specifies the name of the input file to open. It may be given more than once in a single `im_file` module instance. The value must be a [string](#) type [expression](#). For relative filenames you should be aware that NXLog changes its working directory to "/" unless the global `SpoolDir` is set to something else.

On Windows systems the directory separator is the backslash (\). For compatibility reasons the forward slash (/) character can be also used as the directory separator, but this only works for filenames not containing wildcards. If the filename is specified using wildcards, the backslash (\) should be used for the directory separator. Filenames on Windows systems are treated case-insensitively, but case-sensitively on Unix/Linux.

Wildcards are supported in filenames and directories. Wildcards are not regular expressions, but are patterns commonly used by Unix shells to expand filenames (also known as "globbing").

?

Matches a single character only.

*

Matches zero or more characters.

*

Matches the asterisk (*) character.

\?

Matches the question mark (?) character.

[...]

Used to specify a single character. The class description is a list containing single characters and ranges of characters separated by the hyphen (-). If the first character of the class description is ^ or !, the sense of the description is reversed (any character *not* in the list is accepted). Any character can have a backslash (\) preceding it, which is ignored, allowing the characters] and - to be used in the character class, as well as ^ and ! at the beginning.

NOTE

By default, the backslash (\) is used as an escape sequence. Unfortunately this is the same as the directory separator on Windows. Take this into account when specifying wildcarded filenames on this platform. Suppose that log files under the directory C:\test need to be monitored. Specifying the wildcard C:\test*.log will not match because * becomes a literal asterisk and the filename is treated as non-wildcarded. For this reason the directory separator needs to be escaped: C:\test*.log will match our files. C:\\\\test*.log will also work. When specifying the filename using double quotes, this would become C:\\\\test*.log because the backslash is also used as an escape character inside double quoted [string literals](#).

ActiveFiles

This directive specifies the maximum number of files NXLog will actively monitor. If there are modifications to more files in parallel than the value of this directive, then modifications to files above this limit will only get noticed after the [DirCheckInterval](#) (all data should be collected eventually). Typically there are only a few log sources actively appending data to log files, and the rest of the files are dormant after being rotated, so the default value of 10 files should be sufficient in most cases. This directive is also only relevant in case of a wildcarded [File](#) path.

CloseWhenIdle

If set to TRUE, this boolean directive specifies that open input files should be closed as soon as possible after there is no more data to read. Some applications request an exclusive lock on the log file when written or rotated, and this directive can possibly help if the application tries again to acquire the lock. The default is FALSE.

DirCheckInterval

This directive specifies how frequently, in seconds, the module will check the monitored directory for modifications to files and new files in case of a wildcarded [File](#) path. The default is twice the value of the

PollInterval directive (if **PollInterval** is not set, the default is 2 seconds). Fractional seconds may be specified. It is recommended to increase the default if there are many files which cannot be rotated out and the NXLog process is causing high CPU load.

Exclude

This directive can specify a file or a set of files (using wildcards) to be excluded. More than one occurrence of the **Exclude** directive can be specified.

NoEscape

This boolean directive specifies whether the backslash (\) in file paths should be disabled as an escape sequence. This is especially useful for file paths on Windows. By default, **NoEscape** is FALSE (backslash escaping is enabled and the path separator on Windows must be escaped).

OnEOF

This optional block directive can be used to specify a group of statements to execute when a file has been fully read (on end-of-file). Only one **OnEOF** block can be specified per *im_file* module instance. The following directives are used inside this block.

Exec

This mandatory directive specifies the actions to execute after EOF has been detected and the grace period has passed. Like the normal **Exec** directive, the **OnEOF Exec** can be specified as a normal directive or a block directive.

GraceTimeout

This optional directive specifies the time in seconds to wait before executing the actions configured in the **Exec** block or directive. The default is 1 second.

PollInterval

This directive specifies how frequently the module will check for new files and new log entries, in seconds. If this directive is not specified, it defaults to 1 second. Fractional seconds may be specified (**PollInterval 0.5** will check twice every second).

ReadFromLast

This optional boolean directive instructs the module to only read logs which arrived after NXLog was started if the saved position could not be read (for example on first start). When **SavePos** is TRUE and a previously saved position value could be read, the module will resume reading from this saved position. If **ReadFromLast** is FALSE, the module will read all logs from the file. This can result in quite a lot of messages, and is usually not the expected behavior. If this directive is not specified, it defaults to TRUE.

Recursive

If set to TRUE, this boolean directive specifies that input files set with the **File** directive should be searched recursively under sub-directories. For example, `/var/log/error.log` will match `/var/log/apache2/error.log`. Wildcards can be used in combination with **Recursive**: `/var/log/*.log` will match `/var/log/apache2/access.log`. This directive only causes scanning under the given path and does not affect the processing of wildcarded directories: `/var/* qemu/debian.log` will not match `/var/log/libvirt/qemu/debian.log`. The default is FALSE.

RenameCheck

If set to TRUE, this boolean directive specifies that input files should be monitored for possible file rotation via renaming in order to avoid re-reading the file contents. A file is considered to be rotated when NXLog detects a new file whose inode and size matches that of another watched file which has just been deleted. Note that this does not always work correctly and can yield false positives when a log file is deleted and another is added with the same size. The file system is likely to reuse the inode number of the deleted file and thus the module will falsely detect this as a rename/rotation. For this reason the default value of **RenameCheck** is FALSE: renamed files are considered to be new and the file contents will be re-read.

NOTE

It is recommended to use a naming scheme for rotated files so names of rotated files do not match the wildcard and are not monitored anymore after rotation, instead of trying to solve the renaming issue with this directive.

SavePos

If this boolean directive is set to TRUE, the file position will be saved when NXLog exits. The file position will be read from the cache file upon startup. The default is TRUE: the file position will be saved if this directive is not specified. Even if **SavePos** is enabled, it can be explicitly turned off with the global **NoCache** directive.

87.11.2. Functions

The following functions are exported by im_file.

`string file_name()`

Return the name of the currently open file which the log was read from.

`integer record_number()`

Returns the number of processed records (including the current record) of the currently open file since it was opened or truncated.

87.11.3. Examples

Example 447. Forwarding Logs From a File to a Remote Host

This configuration will read from a file and forward messages via TCP. No additional processing is done.

`nxlog.conf`

```
1 <Input messages>
2   Module  im_file
3   File    "/var/log/messages"
4 </Input>
5
6 <Output tcp>
7   Module  om_tcp
8   Host    192.168.1.1
9   Port    514
10 </Output>
11
12 <Route messages_to_tcp>
13   Path    messages => tcp
14 </Route>
```

87.12. File Integrity Monitoring (im_fim)

This module is capable of scanning files and directories and reporting detected changes and deletions. On the first scan, the checksum of each file is recorded. This checksum is then compared to the checksum value calculated during successive scans. The *im_fim* module works on the filesystem level, so it only has access to file information such as ownership and last modification date, and no information about which user made a change.

Files are checked periodically, not in real-time. If there are multiple changes between two scans, only the cumulative effect is logged. For example, if one user modifies a file and another user reverts the changes before the next scan occurs, only the change in modification time is detected.

For real-time monitoring, auditing must be enabled on the host operating system. See the [File Integrity Monitoring](#) chapter in the User Guide for more information.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.12.1. Configuration

The *im_fim* module accepts the following directives in addition to the [common module directives](#). The **File** directive is required.

File

This mandatory directive specifies the name of the input file to scan. It must be a **string** type [expression](#). See the *im_file* **File** directive for more details on how files can be specified. Wildcards are supported. More than one occurrence of the **File** directive can be used.

Digest

This specifies the digest method (hash function) to be used to calculate the checksum. The default is **sha1**. The following message digest methods can be used: **md2**, **md5**, **mdc2**, **rmd160**, **sha**, **sha1**, **sha224**, **sha256**, **sha384**, and **sha512**.

Exclude

This directive can specify a file or a set of files (using wildcards) to be excluded from the scan. More than one occurrence of the **Exclude** directive can be specified.

NoEscape

This boolean directive specifies whether the backslash (\) in file paths should be disabled as an escape sequence. By default, **NoEscape** is FALSE (the path separator on Windows needs to be escaped).

Recursive

If set to TRUE, this boolean directive specifies that files set with the **File** directive should be searched recursively under sub-directories. For example, **/var/log/error.log** will match **/var/log/apache2/error.log**. Wildcards can be used in combination with **Recursive**: **/var/log/*.log** will match **/var/log/apache2/access.log**. This directive only causes scanning under the given path and does not affect the processing of wildcarded directories: **/var/* qemu/debian.log** will not match **/var/log/libvirt/qemu/debian.log**. The default is FALSE.

ScanInterval

This directive specifies how long the module will wait between scans for modifications, in seconds. The default is 86400 seconds (1 day). The value of **ScanInterval** can be set to **0** to disable periodic scanning and instead invoke scans via the [start_scan\(\)](#) procedure.

87.12.2. Procedures

The following procedures are exported by im_fim.

start_scan();

Start the file integrity scan. This could be invoked from the Schedule block, for example.

87.12.3. Fields

The following fields are used by im_fim.

\$raw_event (type: string)

A string containing the **\$EventTime**, **\$Hostname**, **\$EventType**, **\$Object**, and other fields (as applicable) from the event.

\$Digest (type: *string*)

The calculated digest (checksum) value.

\$DigestName (type: *string*)

The name of the digest used to calculate the checksum value (for example, **SHA1**).

\$EventTime (type: *datetime*)

The time when the modification was detected.

\$EventType (type: *string*)

One of the following values: **CHANGE**, **DELETE**, **RENAME**, or **NEW**.

\$FileName (type: *string*)

The name of the file that the changes were detected on.

\$FileSize (type: *integer*)

The size of the file in bytes after the modification.

\$Hostname (type: *string*)

The name of the originating computer.

\$ModificationTime (type: *datetime*)

The modification time (mtime) of the file when the change is detected.

\$Object (type: *string*)

One of the following values: **DIRECTORY** or **FILE**.

\$PrevDigest (type: *string*)

The calculated digest (checksum) value from the previous scan.

\$PrevFileName (type: *string*)

The name of the file from the previous scan.

\$PrevFileSize (type: *integer*)

The size of the file in bytes from the previous scan.

\$PrevModificationTime (type: *datetime*)

The modification time (mtime) of the file from the previous scan.

\$Severity (type: *string*)

The severity name: **WARNING**.

\$SeverityValue (type: *integer*)

The **WARNING** severity level value: **3**.

87.12.4. Examples

Example 448. Periodic File Integrity Monitoring

With this configuration, NXLog will monitor the specified directories recursively. Scans will occur hourly.

nxlog.conf

```

1 <Input fim>
2   Module      im_fim
3   File        "/etc/*"
4   Exclude     "/etc/mtab"
5   File        "/bin/*"
6   File        "/sbin/*"
7   File        "/usr/bin/*"
8   File        "/usr/sbin/*"
9   Recursive   TRUE
10  ScanInterval 3600
11 </Input>
```

Example 449. Scheduled Scan

The *im_fim* module provides a [start_scan\(\)](#) procedure that can be called to invoke the scan. The following configuration sets [ScanInterval](#) to zero to disable periodic scanning and uses a [Schedule](#) block instead to trigger the scan every day at midnight.

nxlog.conf

```

1 <Input fim>
2   Module      im_fim
3   File        "/bin/*"
4   File        "/sbin/*"
5   File        "/usr/bin/*"
6   File        "/usr/sbin/*"
7   Recursive   TRUE
8   ScanInterval 0
9   <Schedule>
10  When    @daily
11  Exec    start_scan();
12 </Schedule>
13 </Input>
```

87.13. HTTP(s) (im_http)

This module can be configured to accept HTTP or HTTPS connections. It expects HTTP POST requests from the client. The event message must be in the request body, and will be available in the [\\$raw_event](#) field. The size of the event message must be indicated with *Content-Length* header. The module will not close the connection while valid requests are received in order to operate in Keep-Alive mode. It will respond with *HTTP/1.1 201 Created* to each valid POST request. This acknowledgment ensures reliable message delivery.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.13.1. Configuration

The *im_http* module accepts the following directives in addition to the [common module directives](#).

ListenAddr

The module will accept connections on this IP address or a DNS hostname. The default is [localhost](#). (This directive is named "Host" in [im_tcp/im_ssl](#) for historical reasons.)

Port

The module instance will listen for incoming connections on this port. The default is port 80.

HTTPSAallowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with an unknown or self-signed certificate. The default value is FALSE: all HTTPS connections must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote HTTPS client. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote HTTPS client.

HTTPSCertFile

This specifies the path of the certificate file to be used for the HTTPS handshake.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the HTTPS handshake.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote HTTPS client. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote HTTPS client.

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

HTTPSRequireCert

This boolean directive specifies that the remote HTTPS client must present a certificate. If set to TRUE and there is no certificate presented during the connection handshake, the connection will be refused. The default value is TRUE: each connection must use a certificate.

HTTPSSSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

HTTPSSSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
HTTPSSSLProtocol TLSv1.2
```

87.13.2. Fields

The following fields are used by im_http.

\$raw_event (*type: string*)

The content received in the POST request.

\$MessageSourceAddress (*type: string*)

The IP address of the remote host.

87.13.3. Examples

Example 450. Receiving Logs over HTTPS

This configuration listens for HTTPS connections from localhost. Received log messages are written to file.

```
nxlog.conf
1 <Input http>
2   Module      im_http
3   ListenAddr  127.0.0.1
4   Port        8888
5   HTTPSCertFile    %CERTDIR%/server-cert.pem
6   HTTPSCertKeyFile  %CERTDIR%/server-key.pem
7   HTTPSCAFile     %CERTDIR%/ca.pem
8   HTTPSRequireCert TRUE
9   HTTPSAllowUntrusted FALSE
10 </Input>
11
12 <Output file>
13   Module      om_file
14   File        'output.log'
15 </Output>
16
17 <Route http_to_file>
18   Path        http => file
19 </Route>
```

87.14. Internal (im_internal)

NXLog produces its own logs about its operations, including errors and debug messages. This module makes it possible to insert those internal log messages into a route. Internal messages can also be generated from the NXLog language using the **log_info()**, **log_warning()**, and **log_error()** procedures.

NOTE

Only messages with log level INFO and above are supported. Debug messages are ignored due to technical reasons. For debugging purposes the direct logging facility should be used: see the global **LogFile** and **LogLevel** directives.

WARNING

One must be careful about the use of the *im_internal* module because it is easy to cause message loops. For example, consider the situation when internal log messages are sent to a database. If the database is experiencing errors which result in internal error messages, then these are again routed to the database and this will trigger further error messages, resulting in a loop. In order to avoid a resource exhaustion, the *im_internal* module will drop its messages when the queue of the next module in the route is full. It is recommended to always put the *im_internal* module instance in a separate route.

NOTE

If internal messages are required in Syslog format, they must be explicitly converted with `pm_transformer` or the `to_syslog_bsd()` procedure of the `xm_syslog` module, because the `$raw_event` field is not generated in Syslog format.

87.14.1. Configuration

The *im_internal* module accepts only the [common module directives](#).

87.14.2. Fields

The following fields are used by im_internal.

\$raw_event (type: string)

The string passed to the `log_info()` or other `log_*` procedure.

\$ErrorCode (type: integer)

The error number provided by the Apache portable runtime library, if an error is logged resulting from an operating system error.

\$EventTime (type: datetime)

The current time.

\$Hostname (type: string)

The hostname where the log was produced.

\$Message (type: string)

The same value as `$raw_event`.

\$ProcessID (type: integer)

The process ID of the NXLog process.

\$Severity (type: string)

The severity name of the event.

\$SeverityValue (type: integer)

Depending on the log level of the internal message, the value corresponding to "debug", "info", "warning", "error", or "critical".

\$SourceName (type: string)

Set to `nxlog`.

87.14.3. Examples

Example 451. Forwarding Internal Messages over Syslog UDP

This configuration collects NXLog internal messages, adds BSD Syslog headers, and forwards via UDP.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input internal>
6     Module im_internal
7 </Input>
8
9 <Output udp>
10    Module om_udp
11    Host   192.168.1.1
12    Port   514
13    Exec   to_syslog_bsd();
14 </Output>
15
16 <Route internal_to_udp>
17     Path   internal => udp
18 </Route>
```

87.15. Kafka (im_kafka)

This module implements an [Apache Kafka](#) consumer for collecting event records from a Kafka topic. See also the [om_kafka](#) module.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.15.1. Configuration

The *im_kafka* module accepts the following directives in addition to the [common module directives](#). The **BrokerList** and **Topic** directives are required.

BrokerList

This mandatory directive specifies the list of Kafka brokers to connect to for collecting logs. The list should include ports and be comma-delimited (for example, `localhost:9092,192.168.88.35:19092`).

Topic

This mandatory directive specifies the Kafka topic to collect records from.

CAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote brokers. **CAFile** is required if **Protocol** is set to **ssl**.

CertFile

This specifies the path of the certificate file to be used for the SSL handshake.

CertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake.

Compression

This directive specifies the compression types to use during transfer. Available types depend on the Kafka library, and should include **none** (the default), **gzip**, **snappy**, and **lz4**.

KeyPass

With this directive, a password can be supplied for the certificate key file defined in **CertKeyFile**. This directive is not needed for passwordless private keys.

Partition

This optional integer directive specifies the topic partition to read from. If this directive is not given, messages are collected from partition 0.

Protocol

This optional directive specifies the protocol to use for connecting to the Kafka brokers. Accepted values include **plaintext** (the default) and **ssl**. If **Protocol** is set to **ssl**, then the **CAFile** directive must also be provided.

87.15.2. Examples

Example 452. Using the im_kafka Module

This configuration collects events from a Kafka cluster using the brokers specified. Events are read from the first partition of the **nxlog** topic.

nxlog.conf

```
1 <Input in>
2   Module      im_kafka
3   BrokerList  localhost:9092,192.168.88.35:19092
4   Topic       nxlog
5   Partition   0
6   Protocol    ssl
7   CAFile      /root/ssl/ca-cert
8   CertFile    /root/ssl/client_debian-8.pem
9   CertKeyFile /root/ssl/client_debian-8.key
10  KeyPass     thisisasecret
11 </Input>
```

87.16. Kernel (im_kernel)

This module collects kernel log messages from the kernel log buffer. This module works on Linux, the BSDs, and macOS.

WARNING

In order for NXLog to read logs from the kernel buffer, it may be necessary to disable the system logger (systemd, klogd, or logd) or configure it to not read events from the kernel.

Special privileges are required for reading kernel logs. For this, NXLog needs to be started as root. With the **User** and **Group** global directives, NXLog can then drop its root privileges while keeping the CAP_SYS_ADMIN capability for reading the kernel log buffer.

NOTE

Unfortunately it is not possible to read from the /proc/kmsg pseudo file for an unprivileged process even if the CAP_SYS_ADMIN capability is kept. For this reason the /proc/kmsg interface is not supported by the *im_kernel* module. The **im_file** module should work fine with the /proc/kmsg pseudo file if one wishes to collect kernel logs this way, though this will require NXLog to be running as root.

Log Sample

```
<6>Some message from the kernel.↔
```

Kernel messages are valid BSD Syslog messages, with a priority from 0 (emerg) to 7 (debug), but do not contain timestamp and hostname fields. These can be parsed with the [xm_syslog parse_syslog_bsd\(\)](#) procedure, and the timestamp and hostname fields will be added by NXLog.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.16.1. Configuration

The `im_kernel` module accepts the following directives in addition to the [common module directives](#).

DeviceFile

This directive sets the device file from which to read events, for non-Linux platforms. If this directive is not specified, the default is `/dev/klog`.

PollInterval

This directive specifies how frequently the module will check for new events, in seconds, on Linux. If this directive is not specified, the default is 1 second. Fractional seconds may be specified (`PollInterval 0.5` will check twice every second).

87.16.2. Examples

Example 453. Reading Messages From the Kernel

This configuration collects log messages from the kernel and writes them to file. This should work on Linux, the BSDs, and macOS (but the system logger may need to be disabled or reconfigured).

`nxlog.conf`

```
1 # Drop privileges after being started as root
2 User nxlog
3 Group nxlog
4
5 <Input kernel>
6   Module im_kernel
7 </Input>
8
9 <Output file>
10  Module om_file
11  File   "tmp/output"
12 </Output>
```

87.17. Linux Audit System (im_linuxaudit)

With this module, NXLog can set up Audit rules and collect the resulting logs directly from the kernel without requiring auditd or other userspace software. If the auditd service is installed, it must not be running.

Rules must be provided using at least one of the [LoadRule](#) and [Rules](#) directives. Rules should be specified using the format documented in the [Defining Persistent Audit Rules](#) section of the Red Hat Enterprise Linux Security Guide.

The `-e` control rule should be included in the ruleset to enable the Audit system (as `-e 1` or `-e 2`). Rules are not automatically removed, either before applying a ruleset or when NXLog exits. To clear the current ruleset before setting rules, begin the ruleset with the `-D` rule. If the Audit configuration is locked when `im_linuxaudit` starts,

NXLog will print a warning and collect events generated by the active ruleset.

87.17.1. Configuration

The `im_linuxaudit` module accepts the following directives in addition to the [common module directives](#). At least one of `LoadRule` and `Rules` must be specified.

`LoadRule`

Use this directive to load a ruleset from an external rules file. This directive can be used more than once. Wildcards can be used to read rules from multiple files.

`Rules`

This directive, specified as a block, can be used to provide Audit rules directly from the NXLog configuration file. The following control rules are supported: `-b`, `-D`, `-e`, `-f`, `-r`, `--loginuid-immutable`, `--backlog_wait_time`, and `--reset-lost`; see `auditctl(8)` for more information.

`Include`

This directive can be used inside a `Rules` block to read rules from a separate file. Like the `LoadRule` directive, wildcards are supported.

`LockConfig`

If this boolean directive is set to TRUE, NXLog will lock the Audit system configuration after the rules have been set. It will not be possible to modify the Audit configuration until after a reboot. The default is FALSE: the Audit configuration will not be locked.

87.17.2. Examples

Example 454. Collecting Audit Logs With LoadRule Directive

This configuration uses a set of external rule files to configure the Audit system.

`nxlog.conf`

```
1 <Input audit>
2   Module      im_linuxaudit
3   LoadRule    'im_linuxaudit_*.rules'
4 </Input>
```

Example 455. Collecting Audit Logs With Rules Block

This configuration lists the rules inside the NXLog configuration file instead of using a separate Audit rules file.

`nxlog.conf`

```
1 <Input audit>
2   Module      im_linuxaudit
3   <Rules>
4     # Watch /etc/passwd for modifications and tag with 'passwd'
5     -w /etc/passwd -p wa -k passwd
6   </Rules>
7 </Input>
```

87.18. Mark (im_mark)

Mark messages are used to indicate periodic activity to assure that the logger is running when there are no log messages coming in from other sources.

By default, if no module-specific directives are set, a log message will be generated every 30 minutes containing `-- MARK --`.

NOTE

The `$raw_event` field is not generated in Syslog format. If mark messages are required in Syslog format, they must be explicitly converted with the `to_syslog_bsd()` procedure.

NOTE

The functionality of the `im_mark` module can be also achieved using the `Schedule` block with a `log_info("--MARK--") Exec` statement, which would insert the messages via the `im_internal` module into a route. Using a single module for this task can simplify configuration.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.18.1. Configuration

The `im_mark` module accepts the following directives in addition to the [common module directives](#).

Mark

This optional directive sets the string for the mark message. The default is `-- MARK --`.

MarkInterval

This optional directive sets the interval for mark messages, in minutes. The default is 30 minutes.

87.18.2. Fields

The following fields are used by `im_mark`.

\$raw_event (type: *string*)

The value defined by the `Mark` directive, `-- MARK --` by default.

\$EventTime (type: *datetime*)

The current time.

\$Message (type: *string*)

The same value as `$raw_event`.

\$ProcessID (type: *integer*)

The process ID of the NXLog process.

\$Severity (type: *string*)

The severity name: `INFO`.

\$SeverityValue (type: *integer*)

The `INFO` severity level value: `2`.

\$SourceName (type: *string*)

Set to `nxlog`.

87.18.3. Examples

Example 456. Using the `im_mark` Module

Here, NXLog will write the specified string to file every minute.

`nxlog.conf`

```
1 <Input mark>
2   Module      im_mark
3   MarkInterval 1
4   Mark        -=| MARK |=- 
5 </Input>
6
7 <Output file>
8   Module      om_file
9   File        "tmp/output"
10 </Output>
11
12 <Route mark_to_file>
13   Path       mark => file
14 </Route>
```

87.19. MS EventLog for Windows XP/2000/2003 (`im_mseventlog`)

This module can be used to collect EventLog messages on Microsoft Windows platforms. The module looks up the available EventLog sources stored under the registry key `SYSTEM\CurrentControlSet\Services\Eventlog` and polls logs from each of these sources or only the sources defined with the `Sources` directive.

Windows Vista, Windows 2008, and later use a new EventLog API which is not backward compatible. Messages in some events produced by sources in this new format cannot be resolved with the old API which is used by this module. If such an event is encountered, a Message similar to the following will be set:

NOTE

The description for EventID XXXX from source SOURCE cannot be read by `_im_mseventlog_` because this does not support the newer WIN2008/Vista EventLog API.

Though the majority of event messages can be read with this module even on Windows 2008/Vista and later, it is recommended to use the `im_msvisalog` module instead.

NOTE

Strings are stored in DLL and executable files and need to be read by the module when reading EventLog messages. If a program (DLL/EXE) is already uninstalled and is not available for looking up a string, the following message will appear instead:

The description for EventID XXXX from source SOURCE cannot be found.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.19.1. Configuration

The `im_mseventlog` module accepts the following directives in addition to the [common module directives](#).

ReadFromLast

This optional boolean directive instructs the module to only read logs which arrived after NXLog was started if

the saved position could not be read (for example on first start). When **SavePos** is TRUE and a previously saved position value could be read, the module will resume reading from this saved position. If **ReadFromLast** is FALSE, the module will read all logs from the EventLog. This can result in quite a lot of messages, and is usually not the expected behavior. If this directive is not specified, it defaults to TRUE.

SavePos

This boolean directive specifies that the file position should be saved when NXLog exits. The file position will be read from the cache file upon startup. The default is TRUE: the file position will be saved if this directive is not specified. Even if **SavePos** is enabled, it can be explicitly turned off with the global **NoCache** directive.

Sources

This optional directive takes a comma-separated list of EventLog filenames, such as **Security**, **Application**, to select specific EventLog sources for reading. If this directive is not specified, then all available EventLog sources are read (as listed in the registry). This directive should not be confused with the **\$SourceName** field contained within the EventLog and it is not a list of such names. The value of this is stored in the **FileName** field.

UTF8

If this optional boolean directive is set to TRUE, all strings will be converted to UTF-8 encoding. Internally this calls the **convert_fields** procedure. The **xm_charconv** module must be loaded for the character set conversion to work. The default is TRUE, but conversion will only occur if the **xm_charconv** module is loaded, otherwise strings will be in the local codepage.

87.19.2. Fields

The following fields are used by im_mseventlog.

\$raw_event (type: *string*)

A string containing the **\$EventTime**, **\$Hostname**, **\$Severity**, and **\$Message** from the event.

\$AccountName (type: *string*)

The username associated with the event.

\$AccountType (type: *string*)

The type of the account. Possible values are: **User**, **Group**, **Domain**, **Alias**, **Well Known Group**, **Deleted Account**, **Invalid**, **Unknown**, and **Computer**.

\$Category (type: *string*)

The category name resolved from CategoryNumber.

\$CategoryNumber (type: *integer*)

The category number, stored as Category in the EventRecord.

\$Domain (type: *string*)

The domain name of the user.

\$EventID (type: *integer*)

The event ID of the EventRecord.

\$EventTime (type: *datetime*)

The TimeGenerated field of the EventRecord.

\$EventTimeWritten (type: *datetime*)

The TimeWritten field of the EventRecord.

\$EventType (type: *string*)

The type of the event, which is a string describing the severity. Possible values are: **ERROR**, **AUDIT_FAILURE**, **AUDIT_SUCCESS**, **INFO**, **WARNING**, and **UNKNOWN**.

\$FileName (type: *string*)

The logfile source of the event (for example, **Security** or **Application**).

\$Hostname (type: *string*)

The host or computer name field of the EventRecord.

\$Message (type: *string*)

The message from the event.

\$RecordNumber (type: *integer*)

The number of the event record.

\$Severity (type: *string*)

The normalized severity name of the event. See **\$SeverityValue**.

\$SeverityValue (type: *integer*)

The normalized severity number of the event, mapped as follows.

Event Log Severity	Normalized Severity
0/Audit Success	2/INFO
0/Audit Failure	4/ERROR
1/Critical	5/CRITICAL
2/Error	4/ERROR
3/Warning	3/WARNING
4/Information	2/INFO
5/Verbose	1/DEBUG

\$SourceName (type: *string*)

The event source which produced the event (the subsystem or application name).

87.19.3. Examples

Example 457. Forwarding EventLogs from a Windows Machine to a Remote Host

This configuration collects Windows EventLog and forwards the messages to a remote host via TCP.

nxlog.conf

```
1 <Input eventlog>
2     Module      im_mseventlog
3 </Input>
4
5 <Output tcp>
6     Module      om_tcp
7     Host        192.168.1.1
8     Port        514
9 </Output>
10
11 <Route eventlog_to_tcp>
12     Path        eventlog => tcp
13 </Route>
```

87.20. MS EventLog for Windows 2008/Vista and Later (im_msvisalog)

This module can be used to collect EventLog messages on Microsoft Windows platforms which support the newer EventLog API (also known as the Crimson EventLog subsystem), namely Windows 2008/Vista and later. See the official Microsoft documentation about [Event Logs](#). The module supports reading all System, Application, and Custom events. It looks up the available channels and monitors events in each unless the [Query](#) and [Channel](#) directives are explicitly defined. Event logs can be collected from remote servers over MSRPC.

NOTE

This module will not work on Windows 2003 and earlier because Windows Vista, Windows 2008, and later use a new EventLog API which is not available in earlier Windows versions. EventLog messages on these platforms can be collected with the [im_mseventlog](#) module.

NOTE

The Windows EventLog subsystem does not support subscriptions to Debug and Analytic channels, thus it is not possible to collect these types of events with this module.

In addition to the standard set of [fields](#) which are listed under the System section, event providers can define their own additional schema which enables logging additional data under the EventData section. The Security log makes use of this new feature and such additional fields can be seen as in the following XML snippet:

```
<EventData>
<Data Name="SubjectUserSid">S-1-5-18</Data>
<Data Name="SubjectUserName">WIN-OUNNPISDHIG$</Data>
<Data Name="SubjectDomainName">WORKGROUP</Data>
<Data Name="SubjectLogonId">0x3e7</Data>
<Data Name="TargetUserSid">S-1-5-18</Data>
<Data Name="TargetUserName">SYSTEM</Data>
<Data Name="TargetDomainName">NT AUTHORITY</Data>
<Data Name="TargetLogonId">0x3e7</Data>
<Data Name="LogonType">5</Data>
<Data Name="LogonProcessName">Advapi</Data>
<Data Name="AuthenticationPackageName">Negotiate</Data>
<Data Name="WorkstationName" />
<Data Name="LogonGuid">{00000000-0000-0000-0000-000000000000}</Data>
<Data Name="TransmittedServices">-</Data>
<Data Name="LmPackageName">-</Data>
<Data Name="KeyLength">0</Data>
<Data Name="ProcessId">0x1dc</Data>
<Data Name="ProcessName">C:\Windows\System32\services.exe</Data>
<Data Name="IpAddress">-</Data>
<Data Name="IpPort">-</Data>
</EventData>
```

NXLog can extract this data when fields are logged using this schema. The values will be available in the fields of the internal NXLog log structure. This is especially useful because there is no need to write pattern matching rules to extract this data from the message. These fields can be used in filtering rules, be written into SQL tables, or be used to trigger actions. The `Exec` directive can be used for filtering:

```
1 <Input in>
2   Module  im_msvisalog
3   Exec    if ($TargetUserName == 'SYSTEM') OR \
4           ($EventType == 'VERBOSE') drop();
5 </Input>
```

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.20.1. Configuration

The `im_msvisalog` module accepts the following directives in addition to the [common module directives](#).

AddPrefix

If this boolean directive is set to TRUE, names of fields parsed from the `<EventData>` portion of the event XML will be prefixed with `EventData..`. For example, `$EventData.SubjectUserName` will be added to the event record instead of `$SubjectUserName`. This directive defaults to FALSE: field names will not be prefixed.

BatchSize

This optional directive can be used to specify the number of event records the EventLog API will pass to the module for processing. Larger sizes may increase throughput. Note that there is a known issue in the Windows EventLog subsystem: when this value is higher than 31 it may fail to retrieve some events on busy systems, returning the error "EvtNext failed with error 1734: The array bounds are invalid." For this reason, increasing this value is not recommended. The default is `31`.

Channel

The name of the Channel to query. If not specified, the module will read from all sources defined in the registry. See the MSDN documentation about [Event Selection](#).

File

This directive can be used to specify a full path to a log file. Log file types that can be used have the following extensions: **.evt**, **.evtx**, and **.etl**. The path of the file must not be quoted (as opposed to **im_file** and **om_file**). If the **File** directive is specified, the **SavePos** and **ReadFromLast** directives will be ignored and the module will read the whole EventLog file. Reading an EventLog file directly is mostly useful for forensics purposes. The **System** log would be read directly with the following:

```
File C:\Windows\System32\winevt\Logs\System.evtx
```

Language

This optional directive specifies a language to use for rendering the events. The language should be given as a hyphen-separated language/region code (for example, **fr-FR** for French). Note that the required language support must be installed on the system. If this directive is not given, the system's default locale is used.

PollInterval

This directive specifies how frequently the module will check for new events, in seconds. If this directive is not specified, the default is 1 second. Fractional seconds may be specified (**PollInterval 0.5** will check twice every second).

Query

This directive specifies the query for pulling only specific EventLog sources. See the MSDN documentation about **Event Selection**. Note that this directive requires a single-line parameter, so multi-line query XML should be specified using line continuation:

```
1 Query <QueryList> \
2   <Query Id='1'> \
3     <Select Path='Security'>*[System/Level=4]</Select> \
4   </Query> \
5 </QueryList>
```

When the **Query** contains an XPath style expression, the **Channel** must also be specified. Otherwise if an XML Query is specified, the **Channel** should not be used.

QueryXML

This directive is the same as the **Query** directive above, except it can be used as a block. Multi-line XML queries can be used without line continuation, and the XML Query can be copied directly from Event Viewer.

```
1 <QueryXML>
2   <QueryList>
3     <Query Id='1'>
4       <Select Path='Security'>*[System/Level=4]</Select>
5     </Query>
6   </QueryList>
7 </QueryXML>
```

ReadFromLast

This optional boolean directive instructs the module to only read logs which arrived after NXLog was started if the saved position could not be read (for example on first start). When **SavePos** is TRUE and a previously saved position value could be read, the module will resume reading from this saved position. If **ReadFromLast** is FALSE, the module will read all logs from the EventLog. This can result in quite a lot of messages, and is usually not the expected behavior. If this directive is not specified, it defaults to TRUE.

RemoteAuthMethod

This optional directive specifies the authentication method to use. Available values are **Default**, **Negotiate**, **Kerberos**, and **NTLM**. When the directive is not specified, **Default** is used, which is actually **Negotiate**.

RemoteDomain

Domain of the user used for authentication when logging on the remote server to collect event logs.

RemotePassword

Password of the user used for authentication when logging on the remote server to collect event logs.

RemoteServer

This optional directive specifies the name of the remote server to collect event logs from. If not specified, the module will collect locally.

RemoteUser

Name of the user used for authentication when logging on the remote server to collect event logs.

ResolveSID

This optional boolean directive specifies that SID values should be resolved to user names in the **\$Message** field. This is FALSE by default: the module will not translate SID values. Windows Event Viewer shows the Message with the SID values resolved, and this must be enabled to get the same output with NXLog.

SavePos

This boolean directive specifies that the file position should be saved when NXLog exits. The file position will be read from the cache file upon startup. The default is TRUE: the file position is saved if this directive is not specified. Even if **SavePos** is enabled, it can be explicitly turned off with the global **NoCache** directive.

TolerateQueryErrors

This boolean directive specifies that *im_msvisalog* should ignore any invalid sources in the query. The default is FALSE: *im_msvisalog* will fail to start if any source is invalid.

87.20.2. Fields

The following fields are used by *im_msvisalog*.

\$raw_event (type: *string*)

A string containing the **\$EventTime**, **\$Hostname**, **\$Severity**, **\$EventID**, and **\$Message** from the event.

\$AccountName (type: *string*)

The username associated with the event.

\$AccountType (type: *string*)

The type of the account. Possible values are: **User**, **Group**, **Domain**, **Alias**, **Well Known Group**, **Deleted Account**, **Invalid**, **Unknown**, and **Computer**.

\$ActivityID (type: *string*)

The ActivityID as stored in EvtSystemActivityID.

\$Category (type: *string*)

The category name resolved from Task.

\$Channel (type: *string*)

The Channel of the event source (for example, **Security** or **Application**).

\$Domain (type: *string*)

The domain name of the user.

\$EventID (type: *integer*)

The event ID from the EvtSystemEventID field.

\$EventTime (type: *datetime*)

The EvtSystemTimeCreated field.

\$EventType (type: *string*)

The type of the event, which is a string describing the severity. This is translated to its string representation from EvtSystemLevel. Possible values are: CRITICAL, ERROR, AUDIT_FAILURE, AUDIT_SUCCESS, INFO, WARNING, and VERBOSE.

\$Hostname (type: *string*)

The EvtSystemComputer field.

\$Keywords (type: *string*)

The value of the Keywords field from EvtSystemKeywords.

\$Message (type: *string*)

The message from the event.

\$Opcode (type: *string*)

The Opcode string resolved from OpcodeValue.

\$OpcodeValue (type: *integer*)

The Opcode number of the event as in EvtSystemOpcode.

\$ProcessID (type: *integer*)

The process identifier of the event producer as in EvtSystemProcessID.

\$ProviderGuid (type: *string*)

The GUI of the event's provider as stored in EvtSystemProviderGuid. This corresponds to the name of the provider stored in the SourceName field.

\$RecordNumber (type: *integer*)

The number of the event record.

\$RelatedActivityID (type: *string*)

The RelatedActivityID as stored in EvtSystemRelatedActivityID.

\$Severity (type: *string*)

The normalized severity name of the event. See [\\$SeverityValue](#).

\$SeverityValue (type: *integer*)

The normalized severity number of the event, mapped as follows.

Event Log Severity	Normalized Severity
0/Audit Success	2/INFO
0/Audit Failure	4/ERROR
1/Critical	5/CRITICAL
2/Error	4/ERROR
3/Warning	3/WARNING
4/Information	2/INFO

Event Log Severity	Normalized Severity
5/Verbose	1/DEBUG

\$SourceName (type: *string*)

The event source which produced the event, from the EvtSystemProviderName field.

\$TaskValue (type: *integer*)

The task number from the EvtSystemTask field.

\$ThreadID (type: *integer*)

The thread identifier of the event producer as in EvtSystemThreadID.

\$UserID (type: *string*)

The SID which resolves to AccountName, stored in EvtSystemUserID.

\$Version (type: *integer*)

The Version number of the event as in EvtSystemVersion.

87.20.3. Examples

Example 458. Forwarding Windows EventLog from Windows to a Remote Host in Syslog Format

This configuration collects Windows EventLog with the specified query. BSD Syslog headers are added and the messages are forwarded to a remote host via TCP.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input eventlog>
6   Module      im_msvisalog
7   <QueryXML>
8     <QueryList>
9       <Query Id='1'>
10      <Select Path='Application'*></Select>
11      <Select Path='Security'>*[Security/Level=4]</Select>
12      <Select Path='System'>*</Select>
13    </Query>
14  </QueryList>
15 </QueryXML>
16 </Input>
17
18 <Output tcp>
19   Module      om_tcp
20   Host        192.168.1.1
21   Port        514
22   Exec        to_syslog_bsd();
23 </Output>
24
25 <Route eventlog_to_tcp>
26   Path        eventlog => tcp
27 </Route>
```

87.21. Null (im_null)

This module does not generate any input, so basically it does nothing. Yet it can be useful for creating a dummy route, for testing purposes, or for [Scheduled](#) NXLog code execution. The *im_null* module accepts only the [common module directives](#). See [this example](#) for usage.

87.22. OCI (im_oci)

This module can read input from an Oracle database.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.22.1. Configuration

The *im_oci* module accepts the following directives in addition to the [common module directives](#). The **DBname**, **Password**, and **UserName** directives are required.

DBname

Name of the database to read the logs from.

Password

Password for authenticating to the database server.

UserName

Username for authenticating to the database server.

ORACLE_HOME

This optional directive specifies the directory of the Oracle installation.

SavePos

This boolean directive specifies that the last row ID should be saved when NXLog exits. The row ID will be read from the cache file upon startup. The default is TRUE: the row ID is saved if this directive is not specified. Even if **SavePos** is enabled, it can be explicitly turned off with the global [NoCache](#) directive.

87.22.2. Examples

Example 459. Reading Logs from an Oracle Database

This configuration will read logs from the specified database and write them to file.

nxlog.conf

```

1 <Input oci>
2   Module      im_oci
3   dbname      //192.168.1.1:1521/orcl
4   username    user
5   password    secret
6   #oracle_home /home/oracle/instantclient_11_2
7 </Input>
8
9 <Output file>
10  Module      om_file
11  File        tmp/output
12 </Output>
13
14 <Route oci_to_file>
15  Path        oci => file
16 </Route>
```

87.23. ODBC (im_odbc)

ODBC is a database independent abstraction layer for accessing databases. This module uses the ODBC API to read data from database tables. There are several ODBC implementations available, and this module has been tested with unixODBC on Linux (available in most major distributions) and Microsoft ODBC on Windows.

Setting up the ODBC data source is not in the scope of this document. Please consult the relevant ODBC guide: the [unixODBC documentation](#) or the [Microsoft ODBC Data Source Administrator guide](#). The data source must be accessible by the user NXLog is running under.

In order to continue reading only new log entries after a restart, the table must contain an auto increment, serial, or timestamp column named *id* in the returned result set. The value of this column is substituted into the **?** contained in the SELECT (see the [SQL](#) directive).

The column names returned in the result set are mapped directly to NXLog field names. The **\$raw_event** field is constructed from the following:

- the EventTime column or the current time if EventTime was not returned in the result set;
- the Hostname column or the hostname of the machine if Hostname was not returned in the result set;
- the Severity column or "INFO" if Severity was not returned in the result set; and
- all other columns prepended as "columnname: columnvalue", each starting on a new line.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.23.1. Configuration

The *im_odbc* module accepts the following directives in addition to the [common module directives](#). The **ConnectionString** directive is required.

ConnectionString

This specifies the connection string containing the ODBC data source name.

IdIsTimestamp

When this directive is set to TRUE, it instructs the module to treat the *id* field as TIMESTAMP type. If this directive is not specified, it defaults to FALSE: the *id* field is treated as an INTEGER/NUMERIC type.

WARNING This configuration directive has been obsoleted in favor of [IdType timestamp](#).

IdType

This directive specifies the type of the *id* field and accepts the following values: [integer](#), [timestamp](#), and [uniqueidentifier](#). If this directive is not specified, it defaults to [integer](#) and the *id* field is treated as an INTEGER/NUMERIC type.

NOTE The *timestamp* type in Microsoft SQL Server is not a real timestamp according to the [documentation](#). Do not set this to [timestamp](#) if you are using *timestamp* with Microsoft SQL Server.

NOTE The Microsoft SQL Server [uniqueidentifier](#) type is only sequential when initialized with the [NEWSEQUENTIALID](#) function. Even then, the IDs are not guaranteed to be sequential in all cases. For more information, see [uniqueidentifier](#) and [NEWSEQUENTIALID](#) on Microsoft Docs.

MaxIdSQL

This directive can be used to specify an SQL select statement for fetching the last record. [MaxIdSQL](#) is required if [ReadFromLast](#) is set to TRUE. The statement must alias the ID column as *maxid* and return at least one row with at least that column.

```
SELECT MAX(RecordNumber) AS maxid FROM logtable
```

PollInterval

This directive specifies how frequently, in seconds, the module will check for new records in the database by executing the SQL SELECT statement. If this directive is not specified, the default is 1 second. Fractional seconds may be specified ([PollInterval 0.5](#) will check twice every second).

ReadFromLast

This boolean directive instructs the module to only read logs that arrived after NXLog was started if the saved position could not be read (for example on first start). When [SavePos](#) is TRUE and a previously saved position value could be read, the module will resume reading from this saved position. If [ReadFromLast](#) is TRUE, the [MaxIdSQL](#) directive must be set. If this directive is not specified, it defaults to FALSE.

SavePos

This boolean directive specifies that the last row id should be saved when NXLog exits. The row id will be read from the cache file upon startup. The default is TRUE: the row id is saved if this directive is not specified. Even if [SavePos](#) is enabled, it can be explicitly turned off with the global [NoCache](#) directive.

SQL

This mandatory parameter sets the SQL statement the module will execute in order to query data from the data source. The select statement must contain a WHERE clause using the column aliased as *id*.

```
SELECT RecordNumber AS id, DateOccured AS EventTime, data AS Message  
FROM logtable WHERE RecordNumber > ?
```

Note that [WHERE RecordNumber > ?](#) is crucial: without this clause the module will read logs in an endless loop. The result set returned by the select must contain this *id* column which is then stored and used for the next query.

87.23.2. Examples

Example 460. Reading from an ODBC Data Source

This example uses ODBC to connect to the `mydb` database and retrieve log messages. The messages are then forwarded to another agent in the NXLog `binary` format.

`nxlog.conf`

```
1 <Input odbc>
2   Module      im_odbc
3   ConnectionString DSN=mssql;database=mydb;
4   SQL          SELECT RecordNumber AS id, \
5                 DateOccured AS EventTime, \
6                 data AS Message \
7                 FROM logtable WHERE RecordNumber > ?
8 </Input>
9
10 <Output tcp>
11   Module      om_tcp
12   Host        192.168.1.1
13   Port        514
14   OutputType  Binary
15 </Output>
```

87.24. Perl (im_perl)

The [Perl programming language](#) is widely used for log processing and comes with a broad set of modules bundled or available from [CPAN](#). Code can be written more quickly in Perl than in C, and code execution is safer because exceptions (`croak/die`) are handled properly and will only result in an unfinished attempt at log processing rather than taking down the whole NXLog process.

This module makes it possible to execute Perl code in an input module to capture and inject event data directly into NXLog. See also the `om_perl` and `xm_perl` modules.

The module will parse the file specified in the `PerlCode` directive when NXLog starts the module. The Perl code must implement the `read_data` subroutine which will be called by the module. To generate event data, the `Log::Nxlog` Perl module must be included, which provides the following methods.

`log_debug(msg)`

Send the message `msg` to the internal logger on DEBUG log level. This method does the same as the `log_debug()` procedure in NXLog.

`log_info(msg)`

Send the message `msg` to the internal logger on INFO log level. This method does the same as the `log_info()` procedure in NXLog.

`log_warning(msg)`

Send the message `msg` to the internal logger on WARNING log level. This method does the same as the `log_warning()` procedure in NXLog.

`log_error(msg)`

Send the message `msg` to the internal logger on ERROR log level. This method does the same as the `log_error()` procedure in NXLog.

`add_input_data(event)`

Pass the event record to the next module instance in the route. Failure to call this method will result in a memory leak.

logdata_new()

Create a new event record. The return value can be used with the *set_field_*()* methods to insert data.

set_field_boolean(event, key, value)

Set the boolean value in the field named *key*.

set_field_integer(event, key, value)

Set the integer value in the field named *key*.

set_field_string(event, key, value)

Set the string value in the field named *key*.

set_read_timer(delay)

Set the timer in seconds to invoke the *read_data* method again.

NOTE

The *set_read_timer()* method should be called in order to invoke *read_data* again. This is typically used for polling data. The *read_data* method must not block.

For the full NXLog Perl API, see the POD documentation in [Nxlog.pm](#). The documentation can be read with `perldoc Log::Nxlog`.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.24.1. Configuration

The *im_perl* module accepts the following directives in addition to the [common module directives](#).

PerlCode

This mandatory directive expects a file containing valid Perl code that implements the *read_data* subroutine. This file is read and parsed by the Perl interpreter.

87.24.2. Examples

Example 461. Using im_perl to Generate Event Data

In this example, logs are generated by a Perl function that increments a counter and inserts it into the generated line.

nxlog.conf

```
1 <Input perl>
2   Module    im_perl
3   PerlCode  modules/input/perl/perl-input.pl
4 </Input>
```

perl-input.pl

```
use strict;
use warnings;

use Log::Nxlog;

my $counter;

sub read_data
{
  my $event = Log::Nxlog::logdata_new();
  $counter //+= 1;
  my $line = "this is a test line ($counter) that should appear in the output";
  $counter++;
  Log::Nxlog::set_field_string($event, 'raw_event', $line);
  Log::Nxlog::add_input_data($event);
  if ( $counter <= 2 )
  {
    Log::Nxlog::set_read_timer(1);
  }
}
```

87.25. Python (im_python)

This module provides support for collecting log data with methods written in the [Python](#) language. The file specified by the [PythonCode](#) directive should contain a [read_data\(\)](#) method which is called by the *im_python* module instance. See also the [xm_python](#) and [om_python](#) modules.

The Python script should import the [nxlog](#) module, and will have access to the following classes and functions.

nxlog.log_debug(msg)

Send the message *msg* to the internal logger at DEBUG log level. This function does the same as the core [log_debug\(\)](#) procedure.

nxlog.log_info(msg)

Send the message *msg* to the internal logger at INFO log level. This function does the same as the core [log_info\(\)](#) procedure.

nxlog.log_warning(msg)

Send the message *msg* to the internal logger at WARNING log level. This function does the same as the core [log_warning\(\)](#) procedure.

nxlog.log_error(msg)

Send the message *msg* to the internal logger at ERROR log level. This function does the same as the core [log_error\(\)](#) procedure.

```
class nxlog.Module()
```

This class will be instantiated by NXLog and passed to the `read_data()` method in the script.

```
logdata_new()
```

This method returns a new LogData event object.

```
set_read_timer(delay)
```

This method sets a trigger for another read after a specified `delay` in seconds (float).

```
class nxlog.LogData()
```

This class represents a Logdata event object.

```
delete_field(name)
```

This method removes the field `name` from the event record.

```
field_names()
```

This method returns a list with the names of all the fields currently in the event record.

```
get_field(name)
```

This method returns the value of the field `name` in the event.

```
post()
```

This method will submit the LogData event to NXLog for processing by the next module in the route.

```
set_field(name, value)
```

This method sets the value of field `name` to `value`.

```
module
```

This attribute is set to the Module object associated with the event.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.25.1. Configuration

The `im_python` module accepts the following directives in addition to the [common module directives](#).

PythonCode

This mandatory directive specifies a file containing Python code. The `im_python` instance will call a `read_data()` function which must accept an [nxlog.Module\(\)](#) object as its only argument.

87.25.2. Examples

Example 462. Using im_python to Generate Event Data

In this example, a Python script is used to read Syslog events from multiple log files bundled in tar archives, which may be compressed. The `parse_syslog()` procedure is also used to parse the events.

NOTE

To avoid re-reading archives, each one should be removed after reading (see the comments in the script) or other similar functionality implemented.

`nxlog.conf`

```

1 <Extension _syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input in>
6   Module      im_python
7   PythonCode  modules/input/python/2_python.py
8   Exec       parse_syslog();
9 </Input>
```

`2_python.py`

```

import os
import tarfile

import nxlog

LOG_DIR = 'modules/input/python/2_logdir'
POLL_INTERVAL = 30

def read_data(module):
    nxlog.log_debug('Checking for new archives')
    for file in os.listdir(LOG_DIR):
        path = os.path.join(LOG_DIR, file)
        nxlog.log_debug("Attempting to read from '{}'".format(path))
        try:
            for line in read_tar(path):
                event = module.logdata_new()
                event.set_field('ImportFile', path)
                event.set_field('raw_event', line)
                event.post()
                nxlog.log_debug("Added event from '{}'".format(path))
            nxlog.log_debug("Added all events from '{}'".format(path))
            # Each archive should be removed after reading to prevent reading
            # the same file again. Requires adequate permissions.
            #nxlog.log_debug("Deleting file '{}'".format(path))
            #os.remove(path)
        except tarfile.ReadError:
            msg = "Skipping invalid tar file '{}'".format(path)
            nxlog.log_error(msg)
    # Check for files again after specified delay
    msg = 'Adding a read event with {} seconds delay'.format(POLL_INTERVAL)
    nxlog.log_debug(msg)
    module.set_read_timer(POLL_INTERVAL)

def read_tar(path):
    """Yield a string for each line in each file in tar file."""
    with tarfile.open(path) as tar:
        for file in tar:
            inner_file = tar.extractfile(file)
            for line in inner_file:
                yield line
```

87.26. Redis (im_redis)

This module can retrieve data stored in a Redis server. The module issues *LPOP* commands using the [Redis Protocol](#) to pull data.

The output counterpart, [om_redis](#), can be used to populate the Redis server with data.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.26.1. Configuration

The *im_redis* module accepts the following directives in addition to the [common module directives](#). The **Host** directive is required.

Host

This mandatory directive specifies the IP address or DNS hostname of the Redis server to connect to.

Command

This optional directive can be used to choose between the **LPOP** and **RPOP** commands. The default **Command** is LPOP if this directive is not specified.

InputType

See the **InputType** directive in the list of common module directives. The default is the **Dgram** reader function, which expects a plain string. To preserve structured data **Binary** can be used, but it must also be set on the other end.

Key

This specifies the **Key** used by the **LPOP** command. The default is **nxlog**.

PollInterval

This directive specifies how frequently the module will check for new data, in seconds. If this directive is not specified, the default is 1 second. Fractional seconds may be specified (**PollInterval 0.5** will check twice every second).

Port

This specifies the port number of the Redis server. The default is port 6379.

87.27. Windows Registry Monitoring (im_regmon)

This module periodically scans the Windows registry and generates event records if a change in the monitored registry entries is detected.

NOTE This module is only available on Windows.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.27.1. Configuration

The *im_regmon* module accepts the following directives in addition to the [common module directives](#). The **RegValue** directive is required.

RegValue

This mandatory directive specifies the name of the registry entry. It must be a **string** type [expression](#). Wildcards are also supported. See the **File** directive of *im_file* for more details on how wildcarded entries can be specified. More than one occurrence of the **RegValue** directive can be specified. The path of the registry entry specified with this directive must start with one of the following: **HKCC**, **HKU**, **HKCU**, **HKCR**, or **HKLM**.

64BitView

If set to TRUE, this boolean directive indicates that the 64 bit registry view should be monitored. The default is TRUE.

Digest

This specifies the digest method (hash function) to be used to calculate the checksum. The default is sha1. The following message digest methods can be used: md2, md5, mdc2, rmd160, sha, sha1, sha224, sha256, sha384, and sha512.

Exclude

This directive specifies a single registry path or a set of registry values (using wildcards) to be excluded from the scan. More than one occurrence of the **Exclude** directive can be used.

Recursive

If set to TRUE, this boolean directive specifies that registry entries set with the **RegValue** directive should be scanned recursively under subkeys. For example, **HKCU\test\value** will match **HKCU\test\subkey\value**. Wildcards can be used in combination with **Recursive**: **HKCU\test\value*** will match **HKCU\test\subkey\value2**. This directive only causes scanning under the given path: **HKCU*\value** will not match **HKCU\test\subkey\value**. The default is FALSE.

ScanInterval

This directive specifies how frequently, in seconds, the module will check the registry entry or entries for modifications. The default is 86400 (1 day). The value of **ScanInterval** can be set to 0 to disable periodic scanning and instead invoke scans via the [start_scan\(\)](#) procedure.

87.27.2. Procedures

The following procedures are exported by im_regmon.

`start_scan();`

Trigger the Windows registry integrity scan. This procedure returns before the scan is finished.

87.27.3. Fields

The following fields are used by im_regmon.

`$raw_event (type: string)`

A string containing the `$EventTime`, `$Hostname`, and other fields.

`$Digest (type: string)`

The calculated digest (checksum) value.

`$DigestName (type: string)`

The name of the digest used to calculate the checksum value (for example, `SHA1`).

`$EventTime (type: datetime)`

The current time.

`$EventType (type: string)`

One of the following values: `CHANGE` or `DELETE`.

`$Hostname (type: string)`

The name of the system where the event was generated.

\$PrevDigest (type: *string*)

The calculated digest (checksum) value from the previous scan.

\$PrevValueSize (type: *integer*)

The size of the registry entry's value from the previous scan.

\$RegistryValueName (type: *string*)

The name of the registry entry where the changes were detected.

\$Severity (type: *string*)

The severity name: **WARNING**.

\$SeverityValue (type: *integer*)

The WARNING severity level value: **3**.

\$ValueSize (type: *integer*)

The size of the registry entry's value after the modification.

87.27.4. Examples

Example 463. Periodic Registry Monitoring

This example monitors the registry entry recursively, and scans every 10 seconds. Messages generated by any detected changes will be written to file in JSON format.

```
nxlog.conf
1 <Extension json>
2     Module      xm_json
3 </Extension>
4
5 <Input regmon>
6     Module      im_regmon
7     RegValue    'HKCU\Software\nxlog\*'
8     ScanInterval 10
9 </Input>
10
11 <Output file>
12     Module      om_file
13     File        'C:\test\regmon.log'
14     Exec        to_json();
15 </Output>
16
17 <Route regmon_to_file>
18     Path        regmon => file
19 </Route>
```

Example 464. Scheduled Registry Scan

The *im_regmon* module provides a [start_scan\(\)](#) procedure that can be called to invoke the scan. The following configuration will trigger the scan every day at midnight.

nxlog.conf

```

1 <Extension json>
2   Module      xm_json
3 </Extension>
4
5 <Input regmon>
6   Module      im_regmon
7   RegValue    'HKCU\Software\*'
8   Exclude     'HKCU\Software\nxlog\*'
9   ScanInterval 0
10  <Schedule>
11    When      @daily
12    Exec      start_scan();
13  </Schedule>
14 </Input>
15
16 <Output file>
17   Module      om_file
18   File       'C:\test\regmon.log'
19   Exec      to_json();
20 </Output>
21
22 <Route dailycheck>
23   Path      regmon => file
24 </Route>
```

87.28. Ruby (im_ruby)

This module provides support for collecting log data with methods written in the [Ruby](#) language. See also the [xm_ruby](#) and [om_ruby](#) modules.

The [Nxlog](#) module provides the following classes and methods.

Nxlog.log_info(msg)

Send the message *msg* to the internal logger at DEBUG log level. This method does the same as the core [log_debug\(\)](#) procedure.

Nxlog.log_debug(msg)

Send the message *msg* to the internal logger at INFO log level. This method does the same as the core [log_info\(\)](#) procedure.

Nxlog.log_warning(msg)

Send the message *msg* to the internal logger at WARNING log level. This method does the same as the core [log_warning\(\)](#) procedure.

Nxlog.log_error(msg)

Send the message *msg* to the internal logger at ERROR log level. This method does the same as the core [log_error\(\)](#) procedure.

class Nxlog.Module

This class will be instantiated by NXLog and passed to the method specified by the [Call](#) directive.

logdata_new()

This method returns a new [LogData](#) object.

set_read_timer(delay)

This method sets a trigger for another read after a specified *delay* in seconds (float).

class Nxlog.LogData

This class represents an event.

field_names()

This method returns an array with the names of all the fields currently in the event record.

get_field(name)

This method returns the value of the field *name* in the event.

post()

This method will submit the event to NXLog for processing by the next module in the route.

set_field(name, value)

This method sets the value of field *name* to *value*.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.28.1. Configuration

The *im_ruby* module accepts the following directives in addition to the [common module directives](#). The **RubyCode** directive is required.

RubyCode

This mandatory directive specifies a file containing Ruby code. The *im_ruby* instance will call the method specified by the **Call** directive. The method must accept an [Nxlog.Module](#) object as its only argument.

Call

This optional directive specifies the Ruby method to call. The default is **read_data**.

87.28.2. Examples

Example 465. Using im_ruby to Generate Events

In this example, events are generated by a simple Ruby method that increments a counter. Because this Ruby method does not set the `$raw_event` field, it would be reasonable to use `to_json()` or some other way to preserve the fields for further processing.

nxlog.conf

```
1 <Input in>
2   Module      im_ruby
3   RubyCode   ./modules/input/ruby/input2.rb
4   Call        read_data
5 </Input>
```

input2.rb

```
$index = 0

def read_data(mod)
  Nxlog.log_debug('Creating new event via input.rb')
  $index += 1
  event = mod.logdata_new
  event.set_field('Counter', $index)
  event.set_field('Message', "This is message #{$index}")
  event.post
  mod.set_read_timer 0.3
end
```

87.29. TLS/SSL (im_ssl)

The `im_ssl` module uses the OpenSSL library to provide an SSL/TLS transport. It behaves like the `im_tcp` module, except that an SSL handshake is performed at connection time and the data is sent over a secure channel. Log messages transferred over plain TCP can be eavesdropped or even altered with a man-in-the-middle attack, while the `im_ssl` module provides a secure log message transport.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.29.1. Configuration

The `im_ssl` module accepts the following directives in addition to the [common module directives](#).

Host

The module will accept connections on this IP address or DNS hostname. The default is `localhost`.

Port

The module will listen for incoming connections on this port number. The default is port 514.

AllowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with an unknown or self-signed certificate. The default value is FALSE: all connections must present a trusted certificate.

CADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL

hashed format.

CAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote socket.

CertFile

This specifies the path of the certificate file to be used for the SSL handshake.

CertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake.

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [CertKeyFile](#). This directive is not needed for passwordless private keys.

CRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format.

CRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote socket.

RequireCert

This boolean value specifies that the remote must present a certificate. If set to TRUE and there is no certificate presented during the connection handshake, the connection will be refused. The default value is TRUE: each connection must use a certificate.

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

87.29.2. Fields

The following fields are used by im_ssl.

\$raw_event (type: *string*)

The received string.

\$MessageSourceAddress (type: *string*)

The IP address of the remote host.

87.29.3. Examples

Example 466. Accepting Binary Logs From Another NXLog Agent

This configuration accepts secured log messages in the NXLog [binary](#) format and writes them to file.

nxlog.conf

```
1 <Input ssl>
2   Module      im_ssl
3   Host        localhost
4   Port        23456
5   CAFile      %CERTDIR%/ca.pem
6   CertFile    %CERTDIR%/client-cert.pem
7   CertKeyFile %CERTDIR%/client-key.pem
8   KeyPass     secret
9   InputType   Binary
10 </Input>
11
12 <Output file>
13   Module      om_file
14   File        "tmp/output"
15 </Output>
16
17 <Route ssl_to_file>
18   Path        ssl => file
19 </Route>
```

87.30. TCP (im_tcp)

This module accepts TCP connections on the configured address and port. It can handle multiple simultaneous connections. The TCP transfer protocol provides more reliable log transmission than UDP. If security is a concern, consider using the [im_ssl](#) module instead.

NOTE

This module provides no access control. Firewall rules can be used to deny connections from certain hosts.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.30.1. Configuration

The *im_tcp* module accepts the following directives in addition to the [common module directives](#).

Host

The module will accept connections on this IP address or DNS hostname. For security, the default listen address is [localhost](#) (the *localhost* loopback address is not accessible from the outside). To receive logs from remote hosts, the address specified here must be accessible. The *any* address [0.0.0.0](#) is commonly used here.

Port

The module will listen for incoming connections on this port number. The default port is 514 if this directive is not specified.

87.30.2. Fields

The following fields are used by im_tcp.

\$raw_event (type: *string*)

The received string.

\$MessageSourceAddress (type: *string*)

The IP address of the remote host.

87.30.3. Examples

Example 467. Using the im_tcp Module

With this configuration, NXLog will listen for TCP connections on port 1514 and write received log messages to file.

nxlog.conf

```
1 <Input tcp>
2   Module  im_tcp
3   Host    0.0.0.0
4   Port    1514
5 </Input>
6
7 <Output file>
8   Module  om_file
9   File    "tmp/output"
10 </Output>
11
12 <Route tcp_to_file>
13   Path    tcp => file
14 </Route>
```

87.31. UDP (im_udp)

This module accepts UDP datagrams on the configured address and port. UDP is the transport protocol of the legacy BSD Syslog as described in RFC 3164, so this module can be particularly useful to receive such messages from older devices which do not support other transports.

NOTE

This module provides no access control. Firewall rules can be used to drop log events from certain hosts.

UDP packets can be dropped by the operating system because the protocol does not guarantee reliable message delivery. It is recommended to use the [TCP](#) or [SSL](#) transport modules instead if message loss is a concern.

NOTE

Though NXLog was designed to minimize message loss even in the case of UDP, adjusting the kernel buffers may reduce the likelihood of UDP message loss on a loaded system. The [Priority](#) directive in the Route block can also help in this situation.

For parsing Syslog messages, see the [pm_transformer](#) module or the [parse_syslog_bsd\(\)](#) procedure of [xm_syslog](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.31.1. Configuration

The *im_udp* module accepts the following directives in addition to the [common module directives](#).

Host

The module will accept messages on this IP address or DNS hostname. The default is **localhost**.

Port

The module will listen for incoming connections on this port number. The default is port 514.

SockBufSize

This optional directive sets the socket buffer size (SO_RCVBUF) to the value specified. If not set, the operating system defaults are used. If UDP packet loss is occurring at the kernel level, setting this to a high value (such as **150000000**) may help. On Windows systems the default socket buffer size is extremely low, and using this option is highly recommended.

UseRecvmmmsg

This boolean directive specifies that the **recvmmmsg()** system call should be used, if available, to receive multiple messages per call to improve performance. The default is TRUE.

87.31.2. Fields

The following fields are used by im_udp.

\$raw_event (type: *string*)

The received string.

\$MessageSourceAddress (type: *string*)

The IP address of the remote host.

87.31.3. Examples

Example 468. Using the im_udp Module

This configuration accepts log messages via UDP and writes them to file.

nxlog.conf

```
1 <Input udp>
2   Module  im_udp
3   Host    192.168.1.1
4   Port    514
5 </Input>
6
7 <Output file>
8   Module  om_file
9   File    "tmp/output"
10 </Output>
11
12 <Route udp_to_file>
13   Path    udp => file
14 </Route>
```

87.32. Unix Domain Socket (im_uds)

This module allows log messages to be received over a Unix domain socket. Unix systems traditionally have a /dev/log or similar socket used by the system logger to accept messages. Applications use the syslog(3) system call to send messages to the system logger.

NOTE

It is recommended to disable [FlowControl](#) when this module is used to collect local Syslog messages from the /dev/log Unix domain socket. Otherwise, if the corresponding Output queue becomes full, the syslog() system call will block in any programs trying to write to the system log and an unresponsive system may result.

For parsing Syslog messages, see the [pm_transformer](#) module or the [parse_syslog_bsd\(\)](#) procedure of [xm_syslog](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.32.1. Configuration

The *im_uds* module accepts the following directives in addition to the [common module directives](#).

UDS

This specifies the path of the Unix domain socket. The default is [/dev/log](#).

UDSType

This directive specifies the domain socket type. Supported values are [dgram](#) and [stream](#). The default is [dgram](#).

InputType

See the [InputType](#) directive in the list of common module directives. This defaults to [dgram](#) if [UDSType](#) is set to [dgram](#) or to [linebased](#) if [UDSType](#) is set to [stream](#).

UDSGroup

Use this directive to set the group ownership for the created socket. By default, this is the group NXLog is running as, (which may be specified by the global [Group](#) directive).

UDSOwner

Use this directive to set the user ownership for the created socket. By default, this is the user NXLog is running as (which may be specified by the global [User](#) directive).

UDSPerms

This directive specifies the permissions to use for the created socket. This must be a four-digit octal value beginning with a zero. By default, universal read/write permissions will be set (octal value [0666](#)).

87.32.2. Examples

Example 469. Using the *im_uds* Module

This configuration will accept logs via the specified socket and write them to file.

nxlog.conf

```
1 <Input uds>
2   Module      im_uds
3   UDS        /dev/log
4   FlowControl False
5 </Input>
```

Example 470. Setting Socket Ownership With *im_uds*

This configuration accepts logs via the specified socket, and also specifies ownership and permissions to use for the socket.

nxlog.conf

```
1 <Input uds>
2   Module    im_uds
3   UDS       /opt/nxlog/var/spool/nxlog/socket
4   UDSOwner  root
5   UDSGroup  adm
6   UDSPerms  0660
7 </Input>
```

87.33. Windows Performance Counters (*im_winperfcount*)

This module periodically retrieves the values of the specified Windows Performance Counters to create an event record. Each event record contains a field for each counter. Each field is named according to the name of the corresponding counter.

NOTE This module is only available on Microsoft Windows.

TIP If performance counters are not working or some counters are missing, it may be necessary to rebuild the performance counter registry settings by running `C:\windows\system32\lodctr.exe /R`. See [How to rebuild performance counters on Windows Vista/Server2008/7/Server2008R2](#) on TechNet for more details, including how to save a backup before rebuilding.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.33.1. Configuration

The *im_winperfcount* module accepts the following directives in addition to the [common module directives](#). The **Counter** directive is required.

Counter

This mandatory directive specifies the name of the performance counter that should be polled, such as `\Memory\Available Bytes`. More than one **Counter** directive can be specified to poll multiple counters at once. Available counter names can be listed with `typeperf -q` (see the [typeperf](#) command reference on Microsoft Docs).

PollInterval

This directive specifies how frequently, in seconds, the module will poll the performance counters. If this directive is not specified, the default is 1 second. Fractional seconds may be specified (`PollInterval 0.5` will check twice every second).

UseEnglishCounters

This optional boolean directive specifies whether to use English counter names. This makes it possible to use the same NXLog configuration across all deployments even if the localization differs. If this directive is not specified it defaults to FALSE (native names will be used).

87.33.2. Fields

The following fields are used by im_winperfcount.

\$raw_event (type: string)

A string containing a header (composed of the `$EventTime` and `$Hostname` fields) followed by a list of key-value pairs for each counter.

\$EventTime (type: datetime)

The current time.

\$Hostname (type: string)

The name of the system where the event was generated.

\$ProcessID (type: integer)

The process ID of the NXLog process.

\$Severity (type: string)

The severity name: `INFO`.

\$SeverityValue (type: integer)

The INFO severity level value: `2`.

\$SourceName (type: string)

Set to `nxlog`.

87.33.3. Examples

Example 471. Polling Windows Performance Counters

With this configuration, NXLog will retrieve the specified counters every 60 seconds. The resulting messages will be written to file in JSON format.

nxlog.conf

```
1 <Extension json>
2   Module      xm_json
3 </Extension>
4
5 <Input counters>
6   Module      im_winperfcount
7   Counter    \Memory\Available Bytes
8   Counter    \Process(_Total)\Working Set
9   PollInterval 60
10 </Input>
11
12 <Output file>
13   Module      om_file
14   File        'C:\test\counter.log'
15   Exec        to_json();
16 </Output>
17
18 <Route perfcount>
19   Path        counters => file
20 </Route>
```

87.34. Windows Management Instrumentation (im_wmi)

This module can be used to collect EventLog messages from Microsoft Windows platforms supporting Windows Management Instrumentation (WMI) mode. The module will poll events from all available event sources. The advantage of this module over [im_mseventlog](#) is that NXLog does not need to be installed on the machine wishing to pull logs from (it can work in agent-less mode). Note that WMI can consume a lot more system resources on the remote Windows server than using [im_mseventlog](#). For the list of EventLog fields see [this MSDN page](#) and the [fields provided by im_wmi](#).

NOTE The *im_wmi* module is currently only available on non-Windows platforms.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.34.1. Configuration

The *im_wmi* module accepts the following directives in addition to the [common module directives](#). The [Host](#), [Password](#), and [Username](#) directives are required.

Host

This mandatory directive specifies the IP address or a DNS hostname the module should connect to.

Password

This mandatory directive specifies the password used for authenticating to the remote host.

Username

This mandatory directive specifies the username used for authenticating to the remote host.

Domain

This specifies the domain used for authenticating to the remote host. The default value is **WORKGROUP**.

Namespace

This specifies the namespace used for authenticating to the remote host. The default is **root\cimv2**.

PollInterval

This directive specifies how frequently the module will check for new events, in seconds. If this directive is not specified it defaults to 5 seconds. Fractional seconds may be specified (**PollInterval 0.5** will check twice every second).

ReadFromLast

This optional boolean directive instructs the module to only read logs which arrived after NXLog was started if the saved position could not be read (for example on first start). When **SavePos** is TRUE and a previously saved position value could be read, the module will resume reading from this saved position. If **ReadFromLast** is FALSE, the module will read all logs from the EventLog. This can result in quite a lot of messages, and is usually not the expected behavior. If this directive is not specified, it defaults to TRUE.

SavePos

This boolean directive specifies that the last record number should be saved when NXLog exits. The last record number will be read from the cache file upon startup. The default is TRUE: the record number is saved if this directive is not specified. Even if **SavePos** is enabled, it can be explicitly turned off with the global **NoCache** directive.

87.34.2. Fields

The following fields are used by im_wmi.

\$raw_event (type: *string*)

A string containing the **\$EventTime**, **\$Hostname**, **\$EventType**, **\$EventID**, **\$AccountName**, and **\$Message** from the event.

\$AccountName (type: *string*)

The username associated with the event.

\$Category (type: *string*)

The category name (CategoryString).

\$CategoryNumber (type: *integer*)

The category number (Category).

\$EventCode (type: *integer*)

The event code (EventCode).

\$EventID (type: *integer*)

The event ID (EventIdentifier). Note that only the EventID/SourceName pair is unique, an event ID can refer to a different event in another source.

\$EventTime (type: *datetime*)

The TimeGenerated field of the EventRecord.

\$EventTimeWritten (type: *datetime*)

The TimeWritten field of the EventRecord.

\$EventType (type: *string*)

The type of the event, which is a string describing the severity (EventType). Possible values are: **ERROR**, **AUDIT_FAILURE**, **AUDIT_SUCCESS**, **INFO**, **WARNING**, and **UNKNOWN**.

\$FileName (type: *string*)

The logfile source of the event (for example, **Security** or **Application**).

\$Hostname (type: *string*)

The ComputerName field of the EventRecord (ComputerName).

\$Message (type: *string*)

The message of the event (Message).

\$RecordNumber (type: *integer*)

The record number of the event (RecordNumber).

\$SeverityValue (type: *integer*)

The severity number of the event.

\$SourceName (type: *string*)

The event source which produced the event (SourceName).

87.34.3. Examples

Example 472. Storing Logs From a Remote Windows System in Per-User Files

This configuration uses WMI to collect Windows EventLog from the specified Windows system. The **File** directive uses an expression to specify a different output file for each user (according to the **\$AccountName** field).

nxlog.conf

```
1 <Input wmi>
2   Module      im_wmi
3   Host        192.168.1.1
4   Username    Administrator
5   Password    secret
6   Domain     WORKGROUP
7 </Input>
8
9 <Output file>
10  Module     om_file
11  File       "/var/log/windows/" + $AccountName + ".log"
12 </Output>
13
14 <Route wmi_to_file>
15  Path        wmi => file
16 </Route>
```

87.35. Windows Event Collector (im_wseventing)

This module can be used to collect Windows EventLog from Microsoft Windows clients that have *Windows Event Forwarding (WEF)* configured. This module takes the role of the collector (Subscription Manager) to accept eventlog records from Windows clients over the **WS-Management** protocol. WS-Eventing is a subset of WS-Management used to forward Windows EventLog.

The [im_mseventlog](#) module requires NXLog to be installed as an agent on the source host. The [im_msvisalog](#) module can be configured to pull Windows EventLog remotely from Windows hosts with a NXLog agent running on Windows. The *im_wseventing* module, however, like [im_wmi](#), can be used on all supported platforms including GNU/Linux systems to remotely collect Windows EventLog without requiring any software to be installed on the source host. Windows clients can be configured through Group Policy to forward EventLog to the system running the *im_wseventing* module, without the need to list each client machine individually in the configuration.

The WS-Eventing protocol and *im_wseventing* support HTTPS using X509 certificates and Kerberos to authenticate and securely transfer EventLog.

NOTE

While there are other products implementing the WS-Eventing protocol (such as IBM WebSphere DataPower), this module was implemented with the primary purpose of collecting and parsing forwarded events from Microsoft Windows. Compatibility with other products has not been assessed.

87.35.1. Setup (certificate based for HTTPS mode)

To set up Windows Event Forwarding over HTTPS the following steps are required:

- X509 certificate generation using either OpenSSL or the Windows certificate manager,
- configuration of the NXLog *im_wseventing* module.
- configuration of Windows Remote Management (WinRM) on each Windows source host,

These steps are covered in greater detail below.

NOTE

We will refer to the host running NXLog with the [im_wseventing](#) module as **server**. Under Windows the **Subscription Manager** refers to the same entity since [im_wsevening](#) is what manages the subscription. We will use the name **client** when referring to the Windows hosts sending the logs using WEF.

The client certificate must have the [X509 v3 Extended Key Usage: TLS Web Server Authentication](#) extension and the server certificate needs the [X509 v3 Extended Key Usage: TLS Web Server Authentication](#) extension. You will likely encounter an error when trying to configure WEF and the connection to the server will fail without these extended key usage attributes. Also make sure that the intended purpose of the certificates are set to [Server Authentication](#) and [Client Authentication](#) respectively.

When generating the certificates please ensure that the CN in the server certificate subject matches the reverse DNS name, otherwise you may get errors in the [Microsoft Windows/Event-ForwardingPlugin/Operational](#) eventlog saying [The SSL certificate contains a common name \(CN\) that does not match the hostname.](#)

Generating the certificates with OpenSSL

If you prefer Windows skip to the next section.

For OpenSSL based certificate generation see the [scripts in our public git repository](#).

Generate the CA certificate and private key:

```
SUBJ="/CN=NXLog-WEF-CA/0=nxlog.org/C=HU/ST=state/L=location"
openssl req -x509 -nodes -newkey rsa:2048 -keyout ca-key.pem -out ca-cert.pem -batch -subj "$SUBJ"
-config gencert.cnf
openssl x509 -outform der -in ca-cert.pem -out ca-cert.crt
```

Generate the client certificate and export it together with the CA in PFX format to be imported into the Windows certificate store:

```
CLIENTSUBJ="/CN=winclient.domain.corp/O=nxlog.org/C=HU/ST=state/L=location"

openssl req -new -newkey rsa:2048 -nodes -keyout client-key.pem -out req.pem -batch -subj
"$CLIENTSUBJ" -config gencert.cnf
openssl x509 -req -days 1024 -in req.pem -CA ca-cert.pem -CAkey ca-key.pem -out client-cert.pem
-set_serial 01 -extensions client_cert -extfile gencert.cnf
openssl pkcs12 -export -out client.pfx -inkey client-key.pem -in client-cert.pem -certfile ca-
cert.pem
```

Generate the server certificate to be used by the [im_wseventing](#) module:

```
SERVERSUBJ="/CN=nxlogserver.domain.corp/O=nxlog.org/C=HU/ST=state/L=location"
openssl req -new -newkey rsa:2048 -nodes -keyout server-key.pem -out req.pem -batch -subj
"$SERVERSUBJ" -config gencert.cnf
openssl x509 -req -days 1024 -in req.pem -CA ca-cert.pem -CAkey ca-key.pem -out server-cert.pem
-set_serial 01 -extensions server_cert -extfile gencert.cnf
openssl x509 -outform der -in server-cert.pem -out server-cert.crt
```

In order to generate the certificates with the correct extensions the following is needed in [gencert.cnf](#):

```
[ server_cert ]
basicConstraints=CA:FALSE
nsCertType = server
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
#crlDistributionPoints=URI:http://127.0.0.1/crl.pem

[ client_cert ]
basicConstraints=CA:FALSE
nsCertType = client
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth
```

NOTE

If you are using an intermediary CA please make sure that the [ca-cert.pem](#) file contains—in correct order—the public part of every issuer's certificate. The easiest way to achieve this is to 'cat' the [pem](#) certificates together.

If you have more complex requirements follow [this guide](#) on how to set up a CA and generate certificates with OpenSSL.

Generating the certificates with the Windows certificate manager

For more information on creating certificates under windows see this document: [Request Certificates by Using the Certificate Request Wizard](#).

Make sure to create the certificates with the required extensions as noted above. Once you have issued the certificates you will need to export the server certificate in PFX format. The PFX must contain the private key also, the password may be omitted. The PFX file can then be converted to the PEM format required by [im_wseventing](#) using [openssl](#):

```
openssl pkcs12 -in server.pfx -nocerts -nodes -out server-key.pem
openssl pkcs12 -in server.pfx -nokeys -nodes -out server-cert.pem
```

You will also need to export the CA certificate (without the private key) the same way and convert it into [ca-cert.pem](#).

Configure NXLog with the `im_wseventing` module

You will need to use `server-key.pem`, `server-cert.pem` and `ca-cert.pem` for the `HTTPSCertKeyFile`, `HTTPSCertFile` and `HTTPSCAFile` respectively.

Optionally you can use the [QueryXML](#) option to filter on specific channels or events.

See the [configuration example](#) below on how your `nxlog.conf` should look.

Once the configuration is complete you may start the `nxlog` service.

Configuring WinRM and WEF

1. Install, configure, and enable Windows Remote Management (WinRM) on each source host.
 - a. Make sure the Windows Remote Management (WS-Management) service is installed, running, and set to *Automatic* startup type.
 - b. If WinRM is not already installed, see these instructions on MSDN: [Installation and Configuration for Windows Remote Management](#).
 - c. Check that the proper client authentication method (Certificate) is enabled for WinRM. Issue the following command:

```
winrm get winrm/config/Client/Auth
```

This should produce the following output

```
Auth
Basic = false
Digest = true
Kerberos = true
Negotiate = true
Certificate = true
CredSSP = true [Source="GPO"]
```

If Certificate authentication is set to *false*, it should be enabled with the following:

```
winrm set winrm/config/client/auth @{Certificate="true"}
```

NOTE Windows Remoting does not support event forwarding over unsecured transport (such as HTTP). Therefore it is recommended to disable the Basic authentication:

```
winrm set winrm/config/client/auth @{Basic="false"}
```

- d. Import the client authentication certificate if you used OpenSSL to generate these. In the Certificate MMC snap-in for the `Local Computer` click `More actions - All Tasks - Import...`. Import the `client.pfx` file. Enter the private key password (if set) and make sure the `Include all extended properties` check-box is selected.

NOTE After importing is completed, open the `Certificates` MMC snap-in, select `Computer account` and double-click on the client certificate to check if the full certificate chain is available and trusted. You may want to move the CA certificate under the `Trusted Root Certification Authorities` in order to make the client certificate trusted.

- e. Grant the `NetworkService` account the proper permissions to access the client certificate using the [Windows HTTP Services Certificate Configuration Tool \(WinHttpCertCfg.exe\)](#) and check that the `NetworkService` account has access to the private key file of the client authentication certificate by running the following command:

```
winhttpcertcfg -l -c LOCAL_MACHINE\my -s <certificate subject name>
```

If NetworkService is not listed in the output, grant it permissions by running the following command:

```
winhttpcertcfg -g -c LOCAL_MACHINE\my -s <certificate subject name> -a NetworkService
```

f. In order to access the *Security EventLog*, the *NetworkService* account needs to be added to the *Event Log Readers* group.

g. Configure the source host security policy to enable event forwarding:

i. Run the **Group Policy** MMC snap-in (**gpedit.msc**) and go to **Computer Configuration > Administrative Templates > Windows Components > Event Forwarding**.

ii. Right-click the **SubscriptionManager** setting and select **Properties > [] Enable the SubscriptionManager setting > and click [Show]** to add a server address.

iii. Add at least one setting that specifies the NXLog collector system. The *SubscriptionManager* Properties window contains an **Explain** tab that describes the syntax for the setting. If you have used the **gencert-server.sh** script it should print the subscription manager string that has the following format:

```
Server=HTTPS://<FQDN of im_wseventing><:port>/wsman/, Refresh=<Refresh interval in seconds>, IssuerCA=<certificate authority certificate thumbprint>
```

An example would be as follows:

```
Server=HTTPS://nxlogserver.domain.corp:5985/wsman/, Refresh=14400, IssuerCA=57F5048548A6A98  
3C3A14DA80E0626E4A462FC04
```

iv. To find the IssuerCA fingerprint, open **MMC**, add the **Certificates** snap-in, select the **Local Computer** account find the Issuing CA certificate. Copy the **Thumbprint** from the **Details** tab. Please make sure to eliminate spaces and the invisible non-breaking space that is before the first character of the thumbprint on Windows 2008.

v. After the *SubscriptionManager* setting has been added, ensure the policy is applied by running:

```
gpupdate /force
```

vi. At this point the WinRM service on the Windows client should connect to NXLog and there should be a connection attempt logged in **nxlog.log** and you should soon start seeing events arriving.

87.35.2. Troubleshooting

WEF is not easy to configure and there may be many things that can go wrong. To troubleshoot WEF you should check the Windows Eventlog under the following channels:

- [Applications and Services Logs/Microsoft Windows/Event-ForwardingPlugin](#)
- [Applications and Services Logs/Microsoft Windows/Windows Remote Management](#)
- [Applications and Services Logs/Microsoft Windows/CAPI2](#)

The CN in the server certificate subject must match the reverse dns, otherwise you may get errors in the **Microsoft Windows/Event-ForwardingPlugin/Operational** eventlog saying **The SSL certificate contains a common name (CN) that does not match the hostname**. Also in that case the WinRM service may want to use a CRL url to download the revocation list. If it cannot check the CRL there will be error messages under [Applications and Services Logs/Microsoft Windows/CAPI2](#) such as this:

```
<Result value="80092013">The revocation function was unable to check  
revocation because the revocation server was offline.</Result>
```

In our experience if the FQDN and the reverse DNS of the server is properly set up it shouldn't fail with the CRL check.

Unfortunately the diagnostic messages in the Windows Eventlog are in some cases rather sloppy. You may see messages such as **The forwarder is having a problem communicating with the subscription manager at address https://nxlog:5985/wsman/. Error code is 42424242 and the Error Message is .** Note the empty error message. Other than guessing you may try looking up the error code on the internet...

If the IssuerCA thumbprint is incorrect or it can't locate the certificate in the certificate store then the above error will be logged in the Windows EventLog with **Error code 2150858882.**

The **Refresh** interval should be set to a higher value (e.g. **Refresh=1200**), in the GPO Subscription Manager settings otherwise the windows client will reconnect too frequently resulting in a lot of connection/disconnection messages in nxlog.log.

The forwarder service may disconnect during the TLS handshake with the following message logged in **nxlog.log**. This is normal as long as there is another connection attempt right after the disconnection:

```
2017-09-28 12:16:01 INFO connection accepted from 10.2.0.161:49381
2017-09-28 12:16:01 ERROR _im_wseventing_ got disconnected during SSL handshake
2017-09-28 12:16:01 INFO connection accepted from 10.2.0.161:49381
```

See the article on Technet titled [Windows Event Forwarding to a workgroup Collector Server](#) for further instructions and troubleshooting tips.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.35.3. Configuration

The *im_wseventing* module accepts the following directives in addition to the [common module directives](#). The **Address** and **ListenAddr** directives are required.

Address

This mandatory directive accepts a URL address. This address is sent to the client to notify it where the events should be sent. For example, **Address https://nxlogserver.domain.corp:5985/wsman.**

ListenAddr

This mandatory directive specifies the address of the interface where the module should listen for client connections. Normally the *any* address **0.0.0.0** is used.

ConnectionRetry

This optional directive specifies the reconnection interval. The default value is **PT60.0S** (60 seconds).

ConnectionRetryTotal

This optional directive specifies the maximum number of reconnection attempts. The default is 5 attempts. If the client exceeds the retry count it will consider the subscription to be stale and will not attempt to reconnect.

Expires

This optional directive can be used to specify a duration after which the subscription will expire, or an absolute time when the subscription will expire. By default, the subscription will never expire.

HeartBeats

Heartbeats are dummy events that do not appear in the output. These are used by the client to signal that

logging is still functional if no events are generated during the specified time period. The default heartbeat value of **PT3600.000S** may be overridden with this optional directive.

HTTPSAallowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with an unknown or self-signed certificate. The default value is FALSE: all HTTPS connections must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote HTTPS client. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote HTTPS client.

HTTPSCertFile

This specifies the path of the certificate file to be used for the HTTPS handshake.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the HTTPS handshake.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote HTTPS client. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote HTTPS client.

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

HTTPSSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

HTTPSSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
HTTPSSLProtocol TLSv1.2
```

MaxElements

This optional directive specifies the maximum number of event records to be batched by the client. If this is not specified the default value is decided by the client.

MaxEnvelopeSize

This optional directive can be used to set a limit on the size of the allowed responses, in bytes. The default size is 153600 bytes. Event records exceeding this size will be dropped by the client and replaced by a drop notification.

MaxTime

This optional directive specifies the maximum amount of time allowed to elapse for the client to batch events. The default value is **PT30.000S** (30 seconds).

Port

This specifies the port on which the module will listen for incoming connections. The default is port 5985.

Query

This directive specifies the query for pulling only specific EventLog sources. See the MSDN documentation about [Event Selection](#). Note that this directive requires a single-line parameter, so multi-line query XML should be specified using line continuation:

```
1 Query <QueryList> \
2   <Query Id='1'> \
3     <Select Path='Security'*[System/Level=4]</Select> \
4   </Query> \
5 </QueryList>
```

QueryXML

This directive is the same as the [Query](#) directive above, except it can be used as a block. Multi-line XML queries can be used without line continuation, and the XML Query can be directly copied from Event Viewer.

```
1 <QueryXML>
2   <QueryList>
3     <Query Id='1'>
4       <Select Path='System'*[Security/Level=4]</Select>
5     </Query>
6   </QueryList>
7 </QueryXML>
```

SubscriptionName

The default value of [NXLog Subscription](#) may be overridden by this optional directive. This name will appear in the client logs.

87.35.4. Examples

Example 473. Collecting Forwarded Events from Windows Hosts

This example Input module instance collects Windows EventLog remotely. Two EventLog queries are specified, the first for hostnames matching `foo*` and the second for other hostnames.

`nxlog.conf`

```
1 <Input wseventing>
2   Module      im_wseventing
3   ListenAddr  0.0.0.0
4   Port        5985
5   Address     https://linux.corp.domain.com:5985/wsman
6   HTTPS CertFile %CERTDIR%/server-cert.pem
7   HTTPS CertKeyFile %CERTDIR%/server-key.pem
8   HTTPS CACertFile %CERTDIR%/ca.pem
9   <QueryXML>
10    <QueryList>
11      <Computer>foo*</Computer>
12      <Query Id="0" Path="Application">
13          <Select Path="Application">*</Select>
14      </Query>
15    </QueryList>
16  </QueryXML>
17  <QueryXML>
18    <QueryList>
19      <Query Id="0" Path="Application">
20          <Select Path="Application">*</Select>
21          <Select Path="Microsoft-Windows-Winsock-AFD/Operational">*</Select>
22          <Select Path="Microsoft-Windows-Wired-AutoConfig/Operational">*</Select>
23          <Select Path="Microsoft-Windows-Wordpad/Admin">*</Select>
24          <Select Path="Windows PowerShell">*</Select>
25      </Query>
26    </QueryList>
27  </QueryXML>
28 </Input>
```

87.36. ZeroMQ (im_zmq)

This module provides message transport over [ZeroMQ](#), a scalable high-throughput messaging library.

The corresponding output module is [om_zmq](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

87.36.1. Configuration

The `im_zmq` module accepts the following directives in addition to the [common module directives](#). The `Address`, `ConnectionType`, `Port`, and `SocketType` directives are required.

`Address`

This directive specifies the ZeroMQ socket address.

`ConnectionType`

This mandatory directive specifies the underlying transport protocol. It may be one of the following: [TCP](#), [PGM](#), or [EPGM](#).

Port

This directive specifies the ZeroMQ socket port.

SocketType

This mandatory directive defines the type of the socket to be used. It may be one of the following: **REQ**, **DEALER**, **SUB**, **XSUB**, or **PULL**. This must be set to **SUB** if **ConnectionType** is set to **PGM** or **EPGM**.

InputType

See the **InputType** directive in the list of common module directives. The default is **Dgram**.

Interface

This directive specifies the ZeroMQ socket interface.

87.36.2. Examples

Example 474. Using the im_zmq Module

This example configuration accepts ZeroMQ messages over TCP and writes them to file.

nxlog.conf

```
1 <Input zmq>
2   Module      im_zmq
3   SocketType  PULL
4   ConnectionType TCP
5   Address     10.0.0.1
6   Port        1415
7 </Input>
8
9 <Output file>
10  Module      om_file
11  File        "/var/log/zmq-messages.log"
12 </Output>
13
14 <Route zmq_to_file>
15  Path        zmq => file
16 </Route>
```

Chapter 88. Processor Modules

Processor modules can be used to process log messages in the log message path between configured Input and Output modules.

88.1. Blocker (pm_blocker)

This module blocks log messages and can be used to simulate a blocked route. When the module blocks the data flow, log messages are first accumulated in the buffers, and then the flow control mechanism pauses the input modules. Using the `block()` procedure, it is possible to programmatically stop or resume the data flow. It can be useful for real-world scenarios as well as testing. See the examples below. When the module starts, the blocking mode is disabled by default (it operates like `pm_null` would).

88.1.1. Configuration

The `pm_blocker` module accepts only the [common module directives](#).

88.1.2. Functions

The following functions are exported by pm_blocker.

`boolean is_blocking()`

Return TRUE if the module is currently blocking the data flow, FALSE otherwise.

88.1.3. Procedures

The following procedures are exported by pm_blocker.

`block(boolean mode);`

When `mode` is TRUE, the module will block. A `block(FALSE)` should be called from a Schedule block or another module, it might not get invoked if the queue is already full.

88.1.4. Examples

Example 475. Using the pm_blocker Module

In this example messages are received over UDP and forwarded to another host via TCP. The log data is forwarded during non-working hours (between 7pm and 8am). During working hours, the data is buffered on the disk.

nxlog.conf

```
1 <Input udp>
2   Module im_udp
3   Host 0.0.0.0
4   Port 1514
5 </Input>
6
7 <Processor buffer>
8   Module pm_buffer
9   # 100 MB disk buffer
10  MaxSize 102400
11  Type disk
12 </Processor>
13
14 <Processor blocker>
15   Module pm_blocker
16   <Schedule>
17     When 0 8 * * *
18     Exec blocker->block(TRUE);
19   </Schedule>
20   <Schedule>
21     When 0 19 * * *
22     Exec blocker->block(FALSE);
23   </Schedule>
24 </Processor>
25
26 <Output tcp>
27   Module om_tcp
28   Host 192.168.1.1
29   Port 1514
30 </Output>
31
32 <Route udp_to_tcp>
33   Path udp => buffer => blocker => tcp
34 </Route>
```

88.2. Buffer (pm_buffer)

Messages received over UDP may be dropped by the operating system if packets are not read from the message buffer fast enough. Some logging subsystems using a small circular buffer can overwrite old logs in the buffer if it is not read, also resulting in loss of log data. Buffering can help in such situations.

The *pm_buffer* module supports disk- and memory-based log message buffering. If both are required, multiple *pm_buffer* instances can be used with different settings. Because a memory buffer can be faster, though its size is limited, combining memory and disk based buffering can be a good idea if buffering is frequently used.

The disk-based buffering mode stores the log message data in chunks. When all the data is successfully forwarded from a chunk, it is then deleted in order to save disk space.

NOTE

Using `pm_buffer` is only recommended when there is a chance of message loss. The built-in flow control in NXLog ensures that messages will not be read by the input module until the output side can send, store, or forward. When reading from files (with `im_file`) or the Windows EventLog (with `im_mseventlog` or `im_msvisalog`) it is rarely necessary to use the `pm_buffer` module unless log rotation is used. During a rotation, there is a possibility of dropping some data while the output module (`im_tcp`, for example) is being blocked.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.2.1. Configuration

The `pm_buffer` module accepts the following directives in addition to the [common module directives](#). The `MaxSize` and `Type` directives are required.

MaxSize

This mandatory directive specifies the size of the buffer in kilobytes.

Type

This directive can be set to either `Mem` or `Disk` to select memory- or disk-based buffering.

Directory

This directory will be used to store the disk buffer file chunks. This is only valid if `Type` is set to `Disk`.

WarnLimit

This directive specifies an optional limit, smaller than `MaxSize`, which will trigger a warning message when reached. The log message will not be generated again until the buffer size drops to half of `WarnLimit` and reaches it again in order to protect against a warning message flood.

88.2.2. Functions

The following functions are exported by `pm_buffer`.

`integer buffer_count()`

Return the number of log messages held in the memory buffer.

`integer buffer_size()`

Return the size of the memory buffer in bytes.

88.2.3. Examples

Example 476. Using a Memory Buffer to Protect Against UDP Message Loss

This configuration accepts log messages via UDP and forwards them via TCP. An intermediate memory-based buffer allows the `im_udp` module instance to continue accepting messages even if the `om_tcp` output stops working (caused by downtime of the remote host or network issues, for example).

`nxlog.conf`

```

1 <Input udp>
2   Module      im_udp
3   Host        0.0.0.0
4   Port        514
5 </Input>
6
7 <Processor buffer>
8   Module      pm_buffer
9   # 1 MB buffer
10  MaxSize    1024
11  Type        Mem
12  # warn at 512k
13  WarnLimit  512
14 </Processor>
15
16 <Output tcp>
17   Module      om_tcp
18   Host        192.168.1.1
19   Port        1514
20 </Output>
21
22 <Route udp_to_tcp>
23   Path        udp => buffer => tcp
24 </Route>
```

88.3. Event Correlator (pm_evcorr)

The `pm_evcorr` module provides event correlation functionality in addition to the already available NXLog language features such as `variables` and `statistical counters` which can be also used for event correlation purposes.

This module was greatly inspired by the Perl based correlation tool [SEC](#). Some of the rules of the `pm_evcorr` module were designed to mimic those available in SEC. This module aims to be a better alternative to SEC with the following advantages:

- The correlation rules in SEC work with the current time. With `pm_evcorr` it is possible to specify a time field which is used for elapsed time calculation making offline event correlation possible.
- SEC uses regular expressions extensively, which can become quite slow if there are many correlation rules. In contrast, this module can correlate pre-processed messages using fields from, for example, the `pattern matcher` and `Syslog` parsers without requiring the use of regular expressions (though these are also available for use by correlation rules). Thus testing conditions can be significantly faster when simple comparison is used instead of regular expression based pattern matching.
- This module was designed to operate on fields, making it possible to correlate structured logs in addition to simple free-form log messages.
- Most importantly, this module is written in C, providing performance benefits (where SEC is written in pure Perl).

The rulesets of this module can use a context. A context is an expression which is evaluated during runtime to a value and the correlation rule is checked in the context of this value. For example, to count the number of failed

logins per user and alert if the failed logins exceed 3 for the user, the `$AccountName` would be used as the context. There is a separate context storage for each correlation rule instance. For global contexts accessible from all rule instances, see [module variables](#) and [statistical counters](#).

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.3.1. Configuration

The `pm_evcoll` module accepts the following directives in addition to the [common module directives](#).

The `pm_evcoll` configuration contains correlation rules which are evaluated for each log message processed by the module. Currently there are five rule types supported by `pm_evcoll`: [Absence](#), [Pair](#), [Simple](#), [Suppressed](#), and [Thresholded](#). These rules are defined in configuration blocks. The rules are evaluated in the order they are defined. For example, a correlation rule can change a state, variable, or field which can be then used by a later rule. [File inclusion](#) can be useful to store correlation rules in a separate file.

Absence

This rule type does the opposite of [Pair](#). When [TriggerCondition](#) evaluates to TRUE, this rule type will wait [Interval](#) seconds for [RequiredCondition](#) to become TRUE. If it does not become TRUE, it executes the statement(s) in the [Exec](#) directive(s).

Context

This optional directive specifies an expression to be used as the context. It must evaluate to a value. Usually a field is specified here.

Exec

One or more **Exec** directives must be specified, each taking a [statement](#) as argument.

NOTE

The evaluation of this Exec is not triggered by a log event; thus it does not make sense to use log data related operations such as accessing fields.

Interval

This mandatory directive takes an integer argument specifying the number of seconds to wait for [RequiredCondition](#) to become TRUE. Its value must be greater than 0. The [TimeField](#) directive is used to calculate time.

RequiredCondition

This mandatory directive takes an expression as argument which must evaluate to a [boolean](#) value. When this evaluates to TRUE after [TriggerCondition](#) evaluated to TRUE within [Interval](#) seconds, the statement(s) in the [Exec](#) directive(s) are NOT executed.

TriggerCondition

This mandatory directive takes an expression as argument which must evaluate to a [boolean](#) value.

Pair

When [TriggerCondition](#) evaluates to TRUE, this rule type will wait [Interval](#) seconds for [RequiredCondition](#) to become TRUE. It then executes the statement(s) in the [Exec](#) directive(s).

Context

This optional directive specifies an expression to be used as the context. It must evaluate to a value. Usually a field is specified here.

Exec

One or more **Exec** directives must be specified, each taking a [statement](#) as argument.

Interval

This directive takes an integer argument specifying the number of seconds to wait for [RequiredCondition](#) to become TRUE. If this directive is **0** or not specified, the rule will wait indefinitely for [RequiredCondition](#) to become TRUE. The [TimeField](#) directive is used to calculate time.

RequiredCondition

This mandatory directive takes an expression as argument which must evaluate to a [boolean](#) value. When this evaluates to TRUE after [TriggerCondition](#) evaluated to TRUE within [Interval](#) seconds, the statement(s) in the [Exec](#) directive(s) are executed.

TriggerCondition

This mandatory directive takes an expression as argument which must evaluate to a [boolean](#) value.

Simple

This rule type is essentially the same as the [Exec](#) directive supported by all modules. Because [Execs](#) are evaluated before the correlation rules, the **Simple** rule was also needed to be able to evaluate a statement as the other rules do, following the rule order. The **Simple** block has one directive also with the same name.

Exec

One or more [Exec](#) directives must be specified, with a [statement](#) as argument.

Stop

This rule will stop evaluating successive rules if the [Condition](#) evaluates to TRUE. The optional [Exec](#) directive will be evaluated in this case.

Condition

This mandatory directive takes an expression as argument which must evaluate to a [boolean](#) value. When it evaluates to TRUE, the correlation rule engine will stop checking any further rules.

Exec

One or more [Exec](#) directives may be specified, each taking a [statement](#) as argument. This will be evaluated when the specified [Condition](#) is satisfied. This directive is optional.

Suppressed

This rule type matches the given condition. If the condition evaluates to TRUE, the statement specified with the [Exec](#) directive is evaluated. The rule will then ignore any log messages for the time specified with [Interval](#) directive. This rule is useful for avoiding creating multiple alerts in a short period when a condition is satisfied.

Condition

This mandatory directive takes an expression as argument which must evaluate to a [boolean](#) value.

Context

This optional directive specifies an expression to be used as the context. It must evaluate to a value. Usually a field is specified here.

Exec

One or more [Exec](#) directives must be specified, each taking a [statement](#) as argument.

Interval

This mandatory directive takes an integer argument specifying the number of seconds to ignore the condition. The [TimeField](#) directive is used to calculate time.

Thresholded

This rule type will execute the statement(s) in the [Exec](#) directive(s) if the [Condition](#) evaluates to TRUE [Threshold](#) or more times during the [Interval](#) specified. The advantage of this rule over the use of [statistical counters](#) is that the time window is dynamic and shifts as log messages are processed.

Condition

This mandatory directive takes an expression as argument which must evaluate to a **boolean** value.

Context

This optional directive specifies an expression to be used as the context. It must evaluate to a value. Usually a field is specified here.

Exec

One or more **Exec** directives must be specified, each taking a **statement** as argument.

Interval

This mandatory directive takes an integer argument specifying a time window for **Condition** to become TRUE. Its value must be greater than 0. The **TimeField** directive is used to calculate time. This time window is dynamic, meaning that it will shift.

Threshold

This mandatory directive takes an integer argument specifying the number of times **Condition** must evaluate to TRUE within the given time **Interval**. When the threshold is reached, the module executes the statement(s) in the **Exec** directive(s).

ContextCleanTime

When a Context is used in the correlation rules, these must be purged from memory after they are expired, otherwise using too many context values could result in a high memory usage. This optional directive specifies the interval between context cleanups, in seconds. By default a **60** second cleanup interval is used if any rules use a Context and this directive is not specified.

TimeField

This specifies the name of the **field** to use for calculating elapsed time, such as **EventTime**. The name of the field must be specified without the leading dollar sign (**\$**). If this parameter is not specified, the current time is assumed. This directive makes it possible to accurately correlate events based on the event time recorded in the logs and to do non-real-time event correlation.

88.3.2. Functions

The following functions are exported by pm_evcoll.

unknown **get_prev_event_data(string field_name)**

When the correlation rule triggers an **Exec**, the data might not be available. This function can be used to retrieve fields of the event that triggered the rule. The field must be specified as a string (for example, **get_prev_event_data("EventTime")**). This is applicable only for the **Pair** and **Absence** rule types.

88.3.3. Examples

Example 477. Correlation Rules

This following configuration sample contains a rule for each type.

nxlog.conf

```
1 <Input filein>
2   Module      im_file
3   File        "modules/processor/evcorr/testinput_evcorr2.txt"
4   Exec      if ($raw_event =~ /(\d\d\d\d-\d\d\d\d \d\d:\d\d:\d\d) (.+)/) { \
5           $EventTime = parsedate($1); \
6           $Message = $2;
```

```
7           $raw_event = $Message;
8       }
9 </Input>
10
11 <Input internal>
12   Module      im_internal
13   Exec        $raw_event = $Message;
14   Exec        $EventTime = 2010-01-01 00:01:00;
15 </Input>
16
17 <Output fileout>
18   Module      om_file
19   File        'tmp/output'
20 </Output>
21
22 <Processor evcorr>
23   Module      pm_evcorr
24   TimeField  EventTime
25
26 <Simple>
27   Exec        if $Message =~ /^simple/ $raw_event = "got simple";
28 </Simple>
29
30 <Suppressed>
31 # Match input event and execute an action list, but ignore the
32 # following matching events for the next $Interval seconds.
33 Condition    $Message =~ /^suppressed/
34 Interval    30
35 Exec        $raw_event = "suppressing..";
36 </Suppressed>
37
38 <Pair>
39 # If TriggerCondition is true, wait Interval seconds for
40 # RequiredCondition to be true and then do the Exec. If Interval is
41 # 0, there is no window on matching.
42 TriggerCondition $Message =~ /^pair-first/
43 RequiredCondition $Message =~ /^pair-second/
44 Interval    30
45 Exec        $raw_event = "got pair";
46 </Pair>
47
48 <Absence>
49 # If TriggerCondition is true, wait Interval seconds for
50 # RequiredCondition to be true. If RequiredCondition does not become
51 # true within the specified interval then do the Exec.
52 TriggerCondition $Message =~ /^absence-trigger/
53 RequiredCondition $Message =~ /^absence-required/
54 Interval    10
55 Exec        log_info("'absence-required' not received within 10 secs");
56 </Absence>
57
58 <Thresholded>
59 # If the number of events exceeds the given threshold within the
60 # interval do the Exec. Same as SingleWithThreshold in SEC.
61 Condition    $Message =~ /^thresholded/
62 Threshold    3
63 Interval    60
64 Exec        $raw_event = "got thresholded";
65 </Thresholded>
66
```

```

67  <Stop>
68  Condition      $EventTime < 2010-01-02 00:00:00
69  Exec           log_debug("got stop");
70  </Stop>
71
72  <Simple>
73  # This will be rewritten only if the previous Stop condition is
74  # FALSE.
75  Exec           $raw_event = "rewritten";
76  </Simple>
77
78 </Processor>
79
80 <Route corr>
81  Path          filein, internal => evcorr => fileout
82 </Route>

```

Input Sample

```

2010-01-01 00:00:00 Not simple<
2010-01-01 00:00:01 suppressed1 - Suppress kicks in, will log 'suppressing...'<
2010-01-01 00:00:10 simple1<
2010-01-01 00:00:12 pair-first - now look for pair-second<
2010-01-01 00:00:13 thresholded1<
2010-01-01 00:00:15 thresholded2<
2010-01-01 00:00:19 simple2<
2010-01-01 00:00:20 thresholded3 - will log 'got thresholded'<
2010-01-01 00:00:21 suppressed2 - suppressed and logged as is<
2010-01-01 00:00:22 pair-second - will log 'got pair'<
2010-01-01 00:00:23 suppressed3 - suppressed and logged as is<
2010-01-01 00:00:25 pair-first<
2010-01-01 00:00:26 absence-trigger<
2010-01-01 00:00:29 absence-required - will not log 'got absence'<
2010-01-01 00:00:46 absence-trigger<
2010-01-01 00:00:56 pair-second - will not log 'got pair' because it is over the interval<
2010-01-01 00:00:57 absence-required - will log an additional 'absence-required not received
within 10 secs'<
2010-01-02 00:00:00 this will be rewritten<
2010-01-02 00:00:10 this too<

```

Output Sample

```

Not simple<
suppressing...<
got simple<
pair-first - now look for pair-second<
thresholded1<
thresholded2<
got simple<
got thresholded<
suppressed2 - suppressed and logged as is<
got pair<
suppressed3 - suppressed and logged as is<
pair-first<
absence-trigger<
absence-required - will not log 'got absence'<
absence-trigger<
pair-second - will not log 'got pair' because it is over the interval<
absence-required - will log an additional 'absence-required not received within 10 secs'<
rewritten<
rewritten<
'absence-required' not received within 10 secs<

```

88.4. Filter (pm_filter)

This is a simple module which forwards log messages if the specified condition is TRUE.

This module has been obsoleted by the NXLog language. Filtering is now possible in any module with a conditional `drop()` procedure in an `Exec` block or directive.

Example 478. Filtering Events With drop()

This statement drops the current event if the `$raw_event` field matches the specified regular expression.

```
1 if $raw_event =~ /^Debug/ drop();
```

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.4.1. Configuration

The `pm_filter` module accepts the following directives in addition to the [common module directives](#).

Condition

This mandatory directive takes an expression as argument which must evaluate to a `boolean` value. If the expression does not evaluate to TRUE, the log message is discarded.

88.4.2. Examples

Example 479. Filtering Messages

This configuration retains only log messages that match one of the regular expressions, all others are discarded.

nxlog.conf

```
1 <Input uds>
2   Module      im_uds
3   UDS        /dev/log
4 </Input>
5
6 <Processor filter>
7   Module      pm_filter
8   Condition   $raw_event =~ /failed/ or $raw_event =~ /error/
9 </Processor>
10
11 <Output file>
12   Module      om_file
13   File        "/var/log/error"
14 </Output>
15
16 <Route uds_to_file>
17   Path        uds => filter => file
18 </Route>
```

88.5. HMAC Message Integrity (pm_hmac)

In order to protect log messages, this module provides cryptographic checksumming on messages using the HMAC algorithm with a specific hash function. Messages protected this way cannot be altered, deleted, or

inserted without detection. A separate verification procedure using the [pm_hmac_check](#) module is necessary for the receiver.

When the module starts, it creates an initial random hash value which is signed with the private key and stored in `$nxlog.hmac_initial` field. As messages pass through the module, it calculates a hash value using the previous hash value, the initial hash value, and the [fields](#) of the log message. This calculated value is added to the log message as a new field called `$nxlog.hmac`, and can be used to later verify the integrity of the message.

WARNING

If the attacker can insert messages at the source, this module will add a HMAC value and the activity will go unnoticed. This method only secures messages that are already protected with a HMAC value.

NOTE

For this method to work more securely, the private key should be protected by a password and the password should not be stored with the key (the configuration file should not contain the password). This will force the agent to prompt for the password when it is started.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.5.1. Configuration

The `pm_hmac` module accepts the following directives in addition to the [common module directives](#). The `CertKeyFile` directive is required.

CertKeyFile

This mandatory directive specifies the path of the private key file to be used to sign the initial hash value.

Fields

This directive accepts a comma-separated list of fields. These fields will be used for calculating the HMAC value. This directive is optional, and the `$raw_event` field will be used if it is not specified.

HashMethod

This directive sets the hash function. The following message digest methods can be used: `md2`, `md5`, `mdc2`, `rmd160`, `sha`, `sha1`, `sha224`, `sha256`, `sha384`, and `sha512`. The default is `md5`.

KeyPass

This specifies the password of the `CertKeyFile`.

88.5.2. Fields

The following fields are used by `pm_hmac`.

`$nxlog.hmac` (type: *string*)

The digest value calculated from the log message fields.

`$nxlog.hmac_initial` (type: *string*)

The initial HMAC value which starts the chain.

`$nxlog.hmac_sig` (type: *string*)

The signature of `nxlog.hmac_initial` created with the private key.

88.5.3. Examples

Example 480. Protecting Messages with a HMAC Value

This configuration uses the `im_uds` module to read log messages from a socket. It then adds a hash value to each message. Finally it forwards them via TCP to another NXLog agent in the `binary` format.

`nxlog.conf`

```

1 <Input uds>
2   Module      im_uds
3   UDS        /dev/log
4 </Input>
5
6 <Processor hmac>
7   Module      pm_hmac
8   CertKeyFile %CERTDIR%/client-key.pem
9   KeyPass    secret
10  HashMethod SHA1
11 </Processor>
12
13 <Output tcp>
14   Module      om_tcp
15   Host        192.168.1.1
16   Port        1514
17   OutputType Binary
18 </Output>
19
20 <Route uds_to_tcp>
21   Path        uds => hmac => tcp
22 </Route>
```

88.6. HMAC Message Integrity Checker (pm_hmac_check)

This module is the pair of `pm_hmac` to check message integrity.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.6.1. Configuration

The `pm_hmac_check` module accepts the following directives in addition to the [common module directives](#). The `CertFile` directive is required.

`CertFile`

This mandatory directive specifies the path of the certificate file to be used to verify the signature of the initial hash value.

`HashMethod`

This directive sets the hash function. The following message digest methods can be used: `md2`, `md5`, `mdc2`, `rmd160`, `sha`, `sha1`, `sha224`, `sha256`, `sha384`, and `sha512`. The default is `md5`. This must be the same as the hash method used for creating the HMAC values.

`CADir`

This optional directive specifies the path to a directory containing certificate authority (CA) certificates, which will be used to verify the certificate. The certificate filenames in this directory must be in the OpenSSL hashed

format.

CAFfile

This optional directive specifies the path of the certificate authority (CA) certificate, which will be used to verify the certificate.

CRLDir

This optional directive specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate. The certificate filenames in this directory must be in the OpenSSL hashed format.

CRLFile

This optional directive specifies the path of the certificate revocation list (CRL), which will be consulted when checking the certificate.

Fields

This directive accepts a comma-separated list of fields. These fields will be used for calculating the HMAC value. This directive is optional, and the **\$raw_event** field will be used if it is not specified.

88.6.2. Fields

The following fields are used by pm_hmac_check.

\$nxlog.hmac (type: *string*)

The HMAC value stored in this field is compared against the calculated value. This field is generated by the pm_hmac module.

\$nxlog.hmac_initial (type: *string*)

The initial HMAC value which starts the chain. This is generated by the pm_hmac module.

\$nxlog.hmac_sig (type: *string*)

The signature of nxlog.hmac_initial to be verified with the certificate's public key. This field is generated by the pm_hmac module.

88.6.3. Examples

Example 481. Verifying Message Integrity

This configuration accepts log messages in the NXLog [binary](#) format. The HMAC values are checked, then the messages are written to file.

nxlog.conf

```

1 <Input tcp>
2   Module      im_tcp
3   Host        192.168.1.1
4   Port        1514
5   InputType   Binary
6 </Input>
7
8 <Processor hmac_check>
9   Module      pm_hmac_check
10  CertFile    %CERTDIR%/client-cert.pem
11  CAFile     %CERTDIR%/ca.pem
12 # CRLFile    %CERTDIR%/crl.pem
13  HashMethod  SHA1
14 </Processor>
15
16 <Output file>
17  Module      om_file
18  File       "/var/log/msg"
19 </Output>
20
21 <Route tcp_to_file>
22  Path        tcp => hmac_check => file
23 </Route>
```

88.7. Message De-Duplicator (pm_norepeat)

This module can be used to filter out repeating messages. Like Syslog daemons, this module checks the previous message against the current. If they match, the current message is dropped. The module waits one second for duplicated messages to arrive. If duplicates are detected, the first message is forwarded, the rest are dropped, and a message containing "last message repeated n times" is sent instead.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.7.1. Configuration

The `pm_norepeat` module accepts the following directives in addition to the [common module directives](#).

CheckFields

This optional directive takes a comma-separated list of field names which are used to compare log messages. Only the fields listed here are compared, the others are ignored. For example, the `$EventTime` field will be different in repeating messages, so this field should not be used in the comparison. If this directive is not specified, the default field to be checked is `$Message`.

88.7.2. Fields

The following fields are used by `pm_norepeat`.

`$raw_event` (type: *string*)

A string containing the `last message repeated n times` message.

\$EventTime (type: *datetime*)

The time of the last event or the current time if EventTime was not present in the last event.

\$Message (type: *string*)

The same value as **\$raw_event**.

\$ProcessID (type: *integer*)

The process ID of the NXLog process.

\$Severity (type: *string*)

The severity name: **INFO**.

\$SeverityValue (type: *integer*)

The INFO severity level value: **2**.

\$SourceName (type: *string*)

Set to **nxlog**.

88.7.3. Examples

Example 482. Filtering Out Duplicated Messages

This configuration reads log messages from the socket. The **\$Hostname**, **\$SourceName**, and **\$Message** fields are used to detect duplicates. Then the messages are written to file.

```
nxlog.conf
1 <Input uds>
2   Module      im_uds
3   UDS        /dev/log
4 </Input>
5
6 <Processor norepeat>
7   Module      pm_norepeat
8   CheckFields Hostname, SourceName, Message
9 </Processor>
10
11 <Output file>
12   Module     om_file
13   File       "/var/log/messages"
14 </Output>
15
16
17 <Route uds_to_file>
18   Path       uds => norepeat => file
19 </Route>
```

88.8. Null (pm_null)

This module does not do any special processing, so basically it does nothing. Yet it can be used with the **Exec** and **Schedule** directives, like any other module.

The *pm_null* module accepts only the [common module directives](#).

See [this example](#) for usage.

88.9. Pattern Matcher (pm_pattern)

This module makes it possible to execute pattern matching with a pattern database file in XML format. The *pm_pattern* module has been replaced by an extension module, [xm_pattern](#), which provides nearly identical functionality with the improved flexibility of an extension module.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.9.1. Configuration

The *pm_pattern* module accepts the following directives in addition to the [common module directives](#).

PatternFile

This mandatory directive specifies the name of the [pattern database file](#).

88.9.2. Fields

The following fields are used by pm_pattern.

\$PatternID (type: *integer*)

The ID number of the pattern which matched the message.

\$PatternName (type: *string*)

The name of the pattern which matched the message.

88.9.3. Examples

Example 483. Using the pm_pattern Module

This configuration reads BSD Syslog messages from the socket, processes the messages with a pattern file, and then writes them to file in JSON format.

nxlog.conf

```

1 <Extension json>
2   Module      xm_json
3 </Extension>
4
5 <Extension syslog>
6   Module      xm_syslog
7 </Extension>
8
9 <Input uds>
10  Module     im_uds
11    UDS       /dev/log
12    Exec      parse_syslog_bsd();
13 </Input>
14
15 <Processor pattern>
16   Module     pm_pattern
17   PatternFile /var/lib(nxlog/patterndb.xml
18 </Processor>
19
20 <Output file>
21   Module     om_file
22   File      "/var/log/out"
23   Exec      to_json();
24 </Output>
25
26 <Route uds_to_file>
27   Path      uds => pattern => file
28 </Route>

```

The following pattern database contains two patterns to match SSH authentication messages. The patterns are under a group named *ssh* which checks whether the `$SourceName` field is `sshd` and only tries to match the patterns if the logs are indeed from `sshd`. The patterns both extract `AuthMethod`, `AccountName`, and `SourceIP4Address` from the log message when the pattern matches the log. Additionally `TaxonomyStatus` and `TaxonomyAction` are set. The second pattern utilizes the `Exec` block, which is evaluated when the pattern matches.

NOTE

For this pattern to work, the logs must be parsed with `parse_syslog()` prior to processing by the `pm_pattern` module (as in the above example), because it uses the `$SourceName` and `$Message` fields.

patterndb.xml

```

<?xml version='1.0' encoding='UTF-8'?>
<patterndb>
  <created>2010-01-01 01:02:03</created>
  <version>42</version>

  <group>
    <name>ssh</name>
    <id>42</id>
    <matchfield>
      <name>SourceName</name>
      <type>exact</type>
      <value>sshd</value>
    </matchfield>

    <pattern>
      <id>1</id>

```

```
<name>ssh auth success</name>

<matchfield>
  <name>Message</name>
  <type>regexp</type>
  <!-- Accepted publickey for nxlogfan from 192.168.1.1 port 4242 ssh2 -->
  <value>^Accepted (\S+) for (\S+) from (\S+) port \d+ ssh2</value>
  <capturedfield>
    <name>AuthMethod</name>
    <type>string</type>
  </capturedfield>
  <capturedfield>
    <name>AccountName</name>
    <type>string</type>
  </capturedfield>
  <capturedfield>
    <name>SourceIP4Address</name>
    <type>string</type>
  </capturedfield>
</matchfield>

<set>
  <field>
    <name>TaxonomyStatus</name>
    <value>success</value>
    <type>string</type>
  </field>
  <field>
    <name>TaxonomyAction</name>
    <value>authenticate</value>
    <type>string</type>
  </field>
</set>
</pattern>

<pattern>
<id>2</id>
<name>ssh auth failure</name>

<matchfield>
  <name>Message</name>
  <type>regexp</type>
  <value>^Failed (\S+) for invalid user (\S+) from (\S+) port \d+ ssh2</value>
  <capturedfield>
    <name>AuthMethod</name>
    <type>string</type>
  </capturedfield>
  <capturedfield>
    <name>AccountName</name>
    <type>string</type>
  </capturedfield>
  <capturedfield>
    <name>SourceIP4Address</name>
    <type>string</type>
  </capturedfield>
</matchfield>

<set>
  <field>
```

```
<name>TaxonomyStatus</name>
<value>failure</value>
<type>string</type>
</field>
<field>
    <name>TaxonomyAction</name>
    <value>authenticate</value>
    <type>string</type>
</field>
</set>

<exec>
    $TestField = 'test';
</exec>
<exec>
    $TestField = $Testfield + ' value' ;
</exec>
</pattern>

</group>

</patterndb>
```

88.10. Message Format Converter (pm_transformer)

The *pm_transformer* module provides parsers for BSD Syslog, IETF Syslog, CSV, JSON, and XML formatted data and can also convert between. This module is now obsoleted by the functions and procedures provided by the following modules: [xm_syslog](#), [xm_csv](#), [xm_json](#), and [xm_xml](#). Using this module can be slightly faster than calling these procedures from an [Exec](#) directive.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.10.1. Configuration

The *pm_transformer* module accepts the following directives in addition to the [common module directives](#). For conversion to occur, the [InputFormat](#) and [OutputFormat](#) directives must be specified.

InputFormat

This directive specifies the input format of the **\$raw_event** field so that it is further parsed into fields. If this directive is not specified, no parsing will be performed.

CSV

Input is parsed as a comma-separated list of values. See [xm_csv](#) for similar functionality. The input fields must be defined by [CSVInputFields](#).

JSON

Input is parsed as JSON. This does the same as the [parse_json\(\)](#) procedure.

syslog_bsd

Same as [syslog_rfc3164](#).

syslog_ietf

Same as [syslog_rfc5424](#).

syslog_rfc3164

Input is parsed in the BSD Syslog format as defined by RFC 3164. This does the same as the

[parse_syslog_bsd\(\)](#) procedure.

syslog_rfc5424

Input is parsed in the IETF Syslog format as defined by RFC 5424. This does the same as the [parse_syslog_ietf\(\)](#) procedure.

XML

Input is parsed as XML. This does the same as the [parse_xml\(\)](#) procedure.

OutputFormat

This directive specifies the output transformation. If this directive is not specified, fields are not converted and **\$raw_event** is left unmodified.

CSV

Output in **\$raw_event** is formatted as a comma-separated list of values. See [xm_csv](#) for similar functionality.

JSON

Output in **\$raw_event** is formatted as JSON. This does the same as the [to_json\(\)](#) procedure.

syslog_bsd

Same as [syslog_rfc3164](#).

syslog_ietf

Same as [syslog_rfc5424](#).

syslog_rfc3164

Output in **\$raw_event** is formatted in the BSD Syslog format as defined by RFC 3164. This does the same as the [to_syslog_bsd\(\)](#) procedure.

syslog_rfc5424

Output in **\$raw_event** is formatted in the IETF Syslog format as defined by RFC 5424. This does the same as the [to_syslog_ietf\(\)](#) procedure.

syslog_snare

Output in **\$raw_event** is formatted in the SNARE Syslog format. This does the same as the [to_syslog_snare\(\)](#) procedure. This should be used in conjunction with the [im_mseventlog](#) or [im_msvisalog](#) module to produce an output compatible with [Snare Agent for Windows](#).

XML

Output in **\$raw_event** is formatted in XML. This does the same as the [to_xml\(\)](#) procedure.

CSVInputFields

This is a comma-separated list of fields which will be set from the input parsed. The field names must have the dollar sign (\$) prepended.

CSVInputFieldTypes

This optional directive specifies the list of types corresponding to the field names defined in [CSVInputFields](#). If specified, the number of types must match the number of field names specified with [CSVInputFields](#). If this directive is omitted, all fields will be stored as [strings](#). This directive has no effect on the fields-to-CSV conversion.

CSVOutputFields

This is a comma-separated list of message fields which are placed in the CSV lines. The field names must have

the dollar sign (\$) prepended.

88.10.2. Examples

Example 484. Using the pm_transformer Module

This configuration reads BSD Syslog messages from file and writes them to another file in CSV format.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input filein>
6   Module      im_file
7   File        "tmp/input"
8 </Input>
9
10 <Processor transformer>
11   Module     pm_transformer
12   InputFormat syslog_rfc3164
13   OutputFormat csv
14   CSVOutputFields $facility, $severity, $timestamp, $hostname, \
15                           $application, $pid, $message
16 </Processor>
17
18 <Output fileout>
19   Module     om_file
20   File       "tmp/output"
21 </Output>
22
23 <Route filein_to_fileout>
24   Path       filein => transformer => fileout
25 </Route>
```

88.11. Timestamping (pm_ts)

This module provides support for the Time-Stamp Protocol as defined in RFC 3161. A cryptographic hash value of the log messages is sent over a HTTP or HTTPS channel to a Time-Stamp Authority server which creates a cryptographic Time-Stamp signature to prove that the message existed at that time and to be able to verify its validity at a later time. This may be mandatory for regulatory compliance, financial transactions, and legal evidence.

A timestamp request is created for each log message received by this module, and the response is appended to the *tsa_response* field. The module does not request the certificate to be included in the response as this would greatly increase the size of the responses. The certificate used by the server for creating timestamps should be saved manually for later verification. The module establishes one HTTP connection to the server for the timestamping by using HTTP Keep-Alive requests and will reconnect if the remote closes the connection.

Since each log message generates a HTTP(S) request to the Time-Stamp server, the message throughput can be greatly affected. It is recommended that only messages of relevant importance are time-stamped through the use of proper filtering rules applied to messages before they reach the *pm_ts* module instance.

NOTE

Creating timestamps in batch mode (requesting one timestamp on multiple messages) is not supported at this time.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

88.11.1. Configuration

The *pm_ts* module accepts the following directives in addition to the [common module directives](#). The [URL](#) directive is required.

URL

This mandatory directive specifies the URL of the Time-Stamp Authority server. The URL must begin with either <http://> for plain HTTP over TCP or <https://> for HTTP over SSL.

Digest

This specifies the digest method (hash function) to be used. The SHA1 hash function is used by default. The following message digest methods can be used: [md2](#), [md5](#), [mdc2](#), [rmd160](#), [sha](#), [sha1](#), [sha224](#), [sha256](#), [sha384](#), and [sha512](#). Note that the Time-Stamp server must support the digest method specified.

Fields

This directive accepts a comma-separated list of fields. These fields will be used for calculating the hash value sent to the TSA server. This directive is optional, and the [\\$raw_event](#) field is used if it is not specified.

HTTPSAAllowUntrusted

This boolean directive specifies that the connection to the Time-Stamp Authority server should be allowed without certificate verification. If set to TRUE, the connection will be allowed even if the server provides an unknown or self-signed certificate. The default value is FALSE: all Time-Stamp Authority servers must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote Time-Stamp Authority server. The certificate filenames in this directory must be in the OpenSSL hashed format. This directive can only be specified if the [URL](#) begins with [https](https://).

HTTPSCAFile

This specifies the path of the certificate of the certificate authority (CA) certificate, which will be used to check the certificate of the remote Time-Stamp Authority server. This directive can only be specified if the [URL](#) begins with [https](https://).

HTTPSCertFile

This specifies the path of the certificate file to be used for the SSL handshake. This directive can only be specified if the [URL](#) begins with [https](https://). If this directive is not specified but the [URL](#) begins with [https](https://), then an anonymous SSL connection is attempted without presenting a client-side certificate.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake. This directive can only be specified if the [URL](#) begins with [https](https://). If this directive is not specified but the [URL](#) begins with [https](https://), then an anonymous SSL connection is attempted without presenting a client-side certificate.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote Time-Stamp Authority server. The certificate filenames in this directory must be in the OpenSSL hashed format. This directive can only be specified if the [URL](#) begins with [https](https://).

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote Time-Stamp Authority server. This directive can only be specified if the [URL](#) begins with [https](#).

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

HTTPSSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

HTTPSSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: [SSLv2](#), [SSLv3](#), [TLSv1](#), [TLSv1.1](#), and [TLSv1.2](#). By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
HTTPSSLProtocol TLSv1.2
```

88.11.2. Fields

The following fields are used by pm_ts.

\$TSAResponse (*type: binary*)

The response for the Time-Stamp request from the server. This does not include the certificate.

88.11.3. Examples

Example 485. Storing Requested Timestamps in a CSV File

With this configuration, NXLog will read BSD Syslog messages from the socket, add timestamps, and then save the messages to file in CSV format.

nxlog.conf

```
1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input uds>
6   Module      im_uds
7   UDS        /dev/log
8   Exec       parse_syslog_bsd();
9 </Input>
10
11 <Processor ts>
12   Module      pm_ts
13   URL        https://tsa-server.com:8080/tsa
14   Digest      md5
15 </Processor>
16
17 <Processor csv>
18   Module      pm_transformer
19   OutputFormat csv
20   CSVOutputFields $facility, $severity, $timestamp, $hostname, \
21                   $application, $pid, $message, $tsa_response
22 </Processor>
23
24 <Output file>
25   Module      om_file
26   File        "/dev/stdout"
27 </Output>
28
29 <Route uds_to_file>
30   Path        uds => ts => csv => file
31 </Route>
```

Chapter 89. Output Modules

Output modules are responsible for writing event log data to various destinations.

89.1. Batched Compression over TCP or SSL (om_batchcompress)

This module uses its own protocol to send batches of log messages to a remote NXLog instance configured with the [im_batchcompress](#) module. The messages are compressed in batches in order to achieve better compression ratios than would be possible individually. The module serializes and sends all fields across the network so that structured data is preserved. It can be configured to send data using SSL for secure and encrypted data transfer. The protocol contains an acknowledgment in order to ensure that the data is received by the remote server. The batch will be resent if the server does not respond with an acknowledgment.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.1.1. Configuration

The *om_batchcompress* module accepts the following directives in addition to the [common module directives](#). The **Host** directive is required.

Host

This specifies the IP address or a DNS hostname the module should connect to.

Port

The module will connect to this port number on the remote host. The default port is 2514.

AllowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE the remote will be able to connect with unknown and self-signed certificates. The default value is FALSE: all connections must present a trusted certificate.

CADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format.

CAFfile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote socket.

CertFile

This specifies the path of the certificate file to be used for the SSL handshake.

CertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake.

CRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format.

CRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote socket.

FlushInterval

The module will send a batch of data to the remote after this amount of time in seconds, unless [FlushLimit](#) is reached first. This defaults to 5 seconds.

FlushLimit

When the number of events in the output buffer reaches the value specified by this directive, the module will compress and send the batch to the remote. This defaults to 500 events. The [FlushInterval](#) directive may trigger sending the batch before this limit is reached if the log volume is low to ensure that data is sent promptly.

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [CertKeyFile](#). This directive is not needed for passwordless private keys.

LocalPort

This optional directive specifies the local port number of the connection. If this is not specified a random high port number will be used, which is not always ideal in firewalled network environments.

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

UseSSL

This boolean directive specifies that SSL transfer mode should be enabled. The default is FALSE.

89.1.2. Examples

Example 486. Sending Logs With om_batchcompress

This configuration forwards logs in compressed batches to a remote NXLog agent over the default port. Batches are sent at least once every two seconds, or more frequently if the buffer reaches 100 events.

nxlog.conf

```
1 <Output batchcompress>
2   Module      om_batchcompress
3   Host        10.0.0.1
4   Port        2514
5   FlushLimit  100
6   FlushInterval 2
7 </Output>
```

89.2. Blocker (om_blocker)

This module is mostly for testing purposes. It will block log messages in order to simulate a blocked route, like when a network transport output module such as [om_tcp](#) blocks because of a network problem.

The [sleep\(\)](#) procedure can also be used for testing by simulating log message delays.

89.2.1. Configuration

The *om_blocker* module accepts only the [common module directives](#).

89.2.2. Examples

Example 487. Testing Buffering With the om_blocker Module

Because the route in this configuration is blocked, this will test the behavior of the configured memory-based buffer.

nxlog.conf

```
1 <Input uds>
2   Module      im_uds
3   UDS        /dev/log
4 </Input>
5
6 <Processor buffer>
7   Module      pm_buffer
8   WarnLimit  512
9   MaxSize    1024
10  Type       Mem
11 </Processor>
12
13 <Output blocker>
14   Module      om_blocker
15 </Output>
16
17 <Route uds_to_blocker>
18   Path       uds => buffer => blocker
19 </Route>
```

89.3. DBI (om_dbi)

The *om_dbi* module allows NXLog to store log data in external databases. This module utilizes the [libdbi](#) database abstraction library, which supports various database engines such as MySQL, PostgreSQL, MSSQL, Sybase, Oracle, SQLite, and Firebird. An INSERT statement can be specified, which will be executed for each log, to insert into any table schema.

NOTE

The [im_dbi](#) and *om_dbi* modules support GNU/Linux only because of the libdbi library. The [im_odbc](#) and [om_odbc](#) modules provide native database access on Windows.

NOTE

libdbi needs [drivers](#) to access the database engines. These are in the libdbd-* packages on Debian and Ubuntu. CentOS 5.6 has a libdbi-drivers RPM package, but this package does not contain any driver binaries under /usr/lib64/dbd. The drivers for both MySQL and PostgreSQL are in libdbi-db-mysql. If these are not installed, NXLog will return a libdbi driver initialization error.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.3.1. Configuration

The *om_dbi* module accepts the following directives in addition to the [common module directives](#).

Driver

This mandatory directive specifies the name of the libdbi driver which will be used to connect to the database. A DRIVER name must be provided here for which a loadable driver module exists under the name [libdbdDRIVER.so](#) (usually under [/usr/lib/dbd/](#)). The MySQL driver is in the [libdbdmysql.so](#) file.

SQL

This directive should specify the INSERT statement to be executed for each log message. The field names (names beginning with \$) will be replaced with the value they contain. [String](#) types will be quoted.

Option

This directive can be used to specify additional driver options such as connection parameters. The manual of the libdbi driver should contain the options available for use here.

89.3.2. Examples

These two examples are for the plain Syslog fields. Other fields generated by parsers, regular expression rules, the [pm_pattern](#) pattern matcher module, or input modules, can also be used. Notably, the [im_msvisalog](#) and [im_mseventlog](#) modules generate different fields than those shown in these examples.

Example 488. Storing Syslog in a PostgreSQL Database

Below is a table schema which can be used to store Syslog data:

```
CREATE TABLE log (
    id serial,
    timestamp timestamp not null,
    hostname varchar(32) default NULL,
    facility varchar(10) default NULL,
    severity varchar(10) default NULL,
    application varchar(10) default NULL,
    message text,
    PRIMARY KEY (id)
);
```

The following configuration accepts log messages via TCP and uses libdbi to insert log messages into the database.

nxlog.conf

```
1 <Extension syslog>
2     Module xm_syslog
3 </Extension>
4
5 <Input tcp>
6     Module im_tcp
7     Port 1234
8     Host 0.0.0.0
9     Exec parse_syslog_bsd();
10 </Input>
11
12 <Output dbi>
13     Module om_db
14     SQL    INSERT INTO log (facility, severity, hostname, timestamp, \
15                             application, message) \
16             VALUES ($SyslogFacility, $SyslogSeverity, $Hostname, '$EventTime', \
17                      $SourceName, $Message)
18     Driver pgsql
19     Option host 127.0.0.1
20     Option username dbuser
21     Option password secret
22     Option dbname logdb
23 </Output>
24
25 <Route tcp_to_dbi>
26     Path  tcp => dbi
27 </Route>
```

Example 489. Storing Logs in a MySQL Database

This configuration reads log messages from the socket and inserts them into a MySQL database.

nxlog.conf

```

1 <Extension syslog>
2   Module      xm_syslog
3 </Extension>
4
5 <Input uds>
6   Module      im_uds
7   UDS        /dev/log
8   Exec       parse_syslog_bsd();
9 </Input>
10
11 <Output dbi>
12   Module      om_dbi
13   SQL         INSERT INTO log (facility, severity, hostname, timestamp, \
14                           application, message) \
15         VALUES ($SyslogFacility, $SyslogSeverity, $Hostname, '$EventTime', \
16                   $SourceName, $Message)
17   Driver      mysql
18   Option      host 127.0.0.1
19   Option      username mysql
20   Option      password mysql
21   Option      dbname logdb
22 </Output>
23
24 <Route uds_to_db>
25   Path        uds => dbi
26 </Route>
```

89.4. Elasticsearch (om_elasticsearch)

This module allows logs to be stored in an Elasticsearch server. It will connect to the [URL](#) specified in the configuration in either plain HTTP or HTTPS mode. The advantage of this module over [om_http](#) is the support for bulk data operations and dynamic indexing. Event data is sent in batches, greatly reducing the latency caused by the HTTP responses and improving Elasticsearch server performance.

NOTE

This module requires the *xm_json* extension module to be loaded in order to convert the payload to JSON. If the [\\$raw_event](#) field does not start with a left curly bracket ([{}](#)), the module will automatically convert the data to JSON.

89.4.1. Server Setup

NXLog uses the [EventTime](#) field to store the timestamp when the logs are parsed as Syslog using [parse_syslog\(\)](#) or read by [im_msvisalog](#) or [im_mseventlog](#). For other sources, make sure [EventTime](#) is set.

The following index template mapping is needed so that Elasticsearch will use the [EventTime](#) field as the timestamp. The format specification is required so that Elasticsearch can correctly parse the [EventTime](#) in the JSON data.

```
$ curl -XPUT localhost:9200/_template/nxlog -d '  
{  
  "template" : "nxlog*",  
  "mappings" : {  
    "_default_" : {  
      "properties": {  
        "EventTime": {  
          "type": "date",  
          "format": "YYYY-MM-dd HH:mm:ss"  
        }  
      }  
    }  
  }  
}'
```

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.4.2. Configuration

The *om_elasticsearch* module accepts the following directives in addition to the [common module directives](#). The [URL](#) directive is required.

URL

This mandatory directive specifies the URL where the module should POST the event data. The URL also indicates whether to operate in plain HTTP or HTTPS mode. If the port number is not explicitly indicated, it defaults to port 80 for HTTP and port 443 for HTTPS. The URL should point to the `_bulk` endpoint, or Elasticsearch will return *400 Bad Request*.

FlushInterval

The module will send a bulk index command to the defined endpoint after this amount of time in seconds, unless [FlushLimit](#) is reached first. This defaults to 5 seconds.

FlushLimit

When the number of events in the output buffer reaches the value specified by this directive, the module will send a bulk index command to the endpoint defined in [URL](#). This defaults to 500 events. The [FlushInterval](#) directive may trigger sending the bulk index request before this limit is reached if the log volume is low to ensure that data is promptly sent to the indexer.

Index

This directive specifies the index to insert the event data into. It must be a [string](#) type expression. If the expression in the [Index](#) directive is not a constant string (it contains functions, field names, or operators), it will be evaluated for each event to be inserted. The default is `nxlog`. Typically, an expression with `strftime()` is used to generate an index name based on the event's time or the current time (for example, `strftime(now(), "nxlog-%Y%m%d")`).

IndexType

This directive specifies the index type to use in the bulk index command. It must be a [string](#) type expression. If the expression in the [IndexType](#) directive is not a constant string (it contains functions, field names, or operators), it will be evaluated for each event to be inserted. The default is `logs`.

HTTPSAallowUntrusted

This boolean directive specifies that the remote connection should be allowed without certificate verification. If set to TRUE, the connection will be allowed even if the remote HTTPS server presents an unknown or self-signed certificate. The default value is FALSE: the remote HTTPS server must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote HTTPS server. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote HTTPS server.

HTTPSCertFile

This specifies the path of the certificate file to be used for the HTTPS handshake.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the HTTPS handshake.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote HTTPS server. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote HTTPS server.

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

SNI

This optional directive specifies the host name used for Server Name Indication (SNI) in HTTPS mode.

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

89.4.3. Examples

Example 490. Sending Logs to an Elasticsearch Server

This configuration reads log messages from file and forwards them to the Elasticsearch server on localhost.

nxlog.conf

```
1 <Extension json>
2   Module      xm_json
3 </Extension>
4
5 <Input file>
6   Module      im_file
7   File        '/var/log/myapp*.log'
8   # Parse log here if needed
9   # $EventTime should be set here
10 </Input>
11
12 <Output es>
13   Module      om_elasticsearch
14   URL        http://localhost:9200/_bulk
15   FlushInterval 2
16   FlushLimit    100
17   # Create an index daily
18   Index        strftime($EventTime, "nxlog-%Y%m%d")
19   # Use the following if you don't have $EventTime set
20   # Index       strftime(now(), "nxlog-%Y%m%d")
21
22   # Set the index type here. Make sure $SourceName has a value!
23   # This can be anything that evaluates to a string, e.g. $Hostname
24   # or $SourceName. Make sure it has a value!
25   Exec         if not defined $SourceName $SourceName = 'unknown';
26   IndexType    $SourceName
27 </Output>
28
29 <Route file_to_es>
30   Path        file => es
31 </Route>
```

89.5. EventDB (om_eventdb)

This custom output module uses libdrizzle to insert log message data into a MySQL database with a special schema. It also supports Unix domain socket connections to the database for faster throughput.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.5.1. Configuration

The *om_eventdb* module accepts the following directives in addition to the [common module directives](#). The **DBname**, **Password**, and **UserName** directives are required along with one of **Host** and **UDS**.

DBname

Name of the database to read the logs from.

Host

This specifies the IP address or a DNS hostname the module should connect to (the hostname of the MySQL server). This directive cannot be used with **UDS**.

Password

Password for authenticating to the database server.

UDS

For Unix domain socket connections, this directive can be used to specify the path of the socket such as `/var/run/mysql.sock`. This directive cannot be used with the [Host](#) and [Port](#) directive.

UserName

Username for authenticating to the database server.

BulkLoad

If set to TRUE, this optional boolean directive instructs the module to use a bulk-loading technique to load data into the database; otherwise traditional INSERT statements are issued to the server. The default is TRUE.

LoadInterval

This directive specifies how frequently bulk loading should occur, in seconds. It can be only used when [BulkLoad](#) is set to TRUE. The default bulk load interval is 20 seconds.

Port

This specifies the port the module should connect to, on which the database is accepting connections. This directive cannot be used with [UDS](#). The default is port 3306.

TempDir

This directive sets a directory where temporary files are written. It can be only used when [BulkLoad](#) is set to TRUE. If this directive is not specified, the default directory is `/tmp`. If the chosen directory does not exist, the module will try to create it.

89.5.2. Examples

Example 491. Storing Logs in an EventDB Database

This configuration accepts log messages via TCP in the NXLog [binary](#) format and inserts them into a database using libdrizzle.

`nxlog.conf`

```
1 <Input tcp>
2   Module      im_tcp
3   Host        localhost
4   Port        2345
5   InputType   Binary
6 </Input>
7
8 <Output eventdb>
9   Module      om_eventdb
10  Host        localhost
11  Port        3306
12  Username    joe
13  Password    secret
14  Ddbname     eventdb_test2
15 </Output>
16
17 <Route tcp_to_eventdb>
18   Path        tcp => eventdb
19 </Route>
```

89.6. Program (om_exec)

This module will execute a program or script on startup and write (pipe) log data to its standard input. Unless **OutputType** is set to something else, only the contents of the **\$raw_event** field are sent over the pipe. The execution of the program or script will terminate when the module is stopped, which usually happens when NXLog exits and the pipe is closed.

NOTE

The program or script is started when NXLog starts and must not exit until the module is stopped. To invoke a program or script for each log message, use **xm_exec** instead.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.6.1. Configuration

The *om_exec* module accepts the following directives in addition to the [common module directives](#). The **Command** directive is required.

Command

This mandatory directive specifies the name of the program or script to be executed.

Arg

This is an optional parameter. **Arg** can be specified multiple times, once for each argument that needs to be passed to the **Command**. Note that specifying multiple arguments with one **Arg** directive, with arguments separated by spaces, will not work (the **Command** will receive it as one argument).

Restart

Restart the process if it exits. There is a one second delay before it is restarted to avoid a denial-of-service when a process is not behaving. Looping should be implemented in the script itself. This directive is only to provide some safety against malfunctioning scripts and programs. This boolean directive defaults to FALSE: the **Command** will not be restarted if it exits.

89.6.2. Examples

Example 492. Piping Logs to an External Program

With this configuration, NXLog will start the specified command, read logs from socket, and write those logs to the standard input of the command.

nxlog.conf

```
1 <Input uds>
2   Module  im_uds
3   UDS      /dev/log
4 </Input>
5
6 <Output someprog>
7   Module  om_exec
8   Command /usr/bin/someprog
9   Arg     -
10 </Output>
11
12 <Route uds_to_someprog>
13   Path    uds => someprog
14 </Route>
```

89.7. File (om_file)

This module can be used to write log messages to a file.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.7.1. Configuration

The *om_file* module accepts the following directives in addition to the [common module directives](#). The **File** directive is required.

File

This mandatory directive specifies the name of the output file to open. It must be a [string type expression](#). If the expression in the **File** directive is not a constant string (it contains functions, field names, or operators), it will be evaluated before each event is written to the file (and after the **Exec** is evaluated). Note that the filename must be quoted to be a valid string literal, unlike in other directives which take a filename argument. For relative filenames, note that NXLog changes its working directory to "/" unless the global **SpoolDir** is set to something else.

Below are three variations for specifying the same output file on a Windows system:

```
File 'C:\logs\logmsg.txt'  
File "C:\\logs\\\\logmsg.txt"  
File 'C:/logs/logmsg.txt'
```

CacheSize

In case of dynamic filenames, a cache can be utilized to keep files open. This increases performance by reducing the overhead caused by many open/close operations. It is recommended to set this to the number of expected files to be written. Note that this should not be set to more than the number of open files allowed by the system. This caching provides performance benefits on Windows only. Caching is disabled by default.

CreateDir

If set to TRUE, this optional boolean directive instructs the module to create the output directory before opening the file for writing if it does not exist. The default is FALSE.

OutputType

See the [OutputType](#) directive in the list of common module directives. If this directive is not specified the default is [LineBased](#) (the module will use CRLF as the record terminator on Windows, or LF on Unix).

Sync

This optional boolean directive instructs the module to sync the file after each log message is written, ensuring that it is really written to disk from the buffers. Because this can hurt performance, the default is FALSE.

Truncate

This optional boolean directive instructs the module to truncate the file before each write, causing only the most recent log message to be saved. The default is FALSE: messages are appended to the output file.

89.7.2. Functions

The following functions are exported by om_file.

string file_name()

Return the name of the currently open file which was specified using the [File](#) directive. Note that this will be the old name if the filename changes dynamically; for the new name, use the expression specified for the [File](#) directive instead of using this function.

`integer file_size()`

Return the size of the currently open output file in bytes. Returns `undef` if the file is not open. This can happen if [File](#) is not a string literal expression and there was no log message.

89.7.3. Procedures

The following procedures are exported by `om_file`.

`reopen();`

Reopen the current file. This procedure should be called if the file has been removed or renamed, for example with the [file_cycle\(\)](#), [file_remove\(\)](#), or [file_rename\(\)](#) procedures of the `xm_fileop` module. This does not need to be called after [rotate_to\(\)](#) because that procedure reopens the file automatically.

`rotate_to(string filename);`

Rotate the current file to the *filename* specified. The module will then open the original file specified with the [File](#) directive. Note that the `rename(2)` system call is used internally which does not support moving files across different devices on some platforms. If this is a problem, first rotate the file on the same device. Then use the `xm_exec` [exec_async\(\)](#) procedure to copy it to another device or file system, or use the `xm_fileop` [file_copy\(\)](#) procedure.

89.7.4. Examples

Example 493. Storing Raw Syslog Messages into a File

This configuration reads log messages from socket and writes the messages to file. No additional processing is done.

nxlog.conf

```
1 <Input uds>
2   Module im_uds
3   UDS    /dev/log
4 </Input>
5
6 <Output file>
7   Module om_file
8   File   "/var/log/messages"
9 </Output>
10
11 <Route uds_to_file>
12   Path   uds => file
13 </Route>
```

Example 494. File Rotation Based on Size

With this configuration, NXLog accepts log messages via TCP and parses them as BSD Syslog. A separate output file is used for log messages from each host. When the output file size exceeds 15 MB, it will be automatically rotated and compressed.

nxlog.conf

```

1 <Extension exec>
2     Module xm_exec
3 </Extension>
4
5 <Extension syslog>
6     Module xm_syslog
7 </Extension>
8
9 <Input tcp>
10    Module im_tcp
11    Port 1514
12    Host 0.0.0.0
13    Exec parse_syslog_bsd();
14 </Input>
15
16 <Output file>
17     Module om_file
18     File   "tmp/output_" + $Hostname + "_" + month(now())
19     <Exec>
20         if file->file_size() > 15M
21         {
22             $newfile = "tmp/output_" + $Hostname + "_" +
23                         strftime(now(), "%Y%m%d%H%M%S");
24             file->rotate_to($newfile);
25             exec_async("/bin/bzip2", $newfile);
26         }
27     </Exec>
28 </Output>
29
30 <Route tcp_to_file>
31     Path  tcp => file
32 </Route>
```

89.8. HTTP(s) (om_http)

This module will connect to the specified [URL](#) in either plain HTTP or HTTPS mode. Each event is transferred in a single POST request. The module then waits for a response containing a successful status code (200, 201, or 202). It will reconnect and retry the delivery if the remote has closed the connection or a timeout is exceeded while waiting for the response. This HTTP-level acknowledgment ensures that no messages are lost during transfer.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.8.1. Configuration

The *om_http* module accepts the following directives in addition to the [common module directives](#). The [URL](#) directive is required.

URL

This mandatory directive specifies the URL where the module should POST the event data. The module

operates in plain HTTP or HTTPS mode depending on the URL provided, and connects to the hostname specified in the URL. If the port number is not explicitly indicated in the URL, it defaults to port 80 for HTTP and port 443 for HTTPS.

AddHeader

This optional directive specifies an additional header to be added to each HTTP request.

ContentType

This directive sets the *Content-Type* HTTP header to the string specified. The *Content-Type* is set to `text/plain` by default.

HTTPSAllowUntrusted

This boolean directive specifies that the connection should be allowed without certificate verification. If set to TRUE, the connection will be allowed even if the remote HTTPS server presents an unknown or self-signed certificate. The default value is FALSE: the remote HTTPS server must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote HTTPS server. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote HTTPS server.

HTTPSCertFile

This specifies the path of the certificate file to be used for the HTTPS handshake.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the HTTPS handshake.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote HTTPS server. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL) which will be consulted when checking the certificate of the remote HTTPS server.

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

HTTPSSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

HTTPSSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: `SSLv2`, `SSLv3`, `TLSv1`, `TLSv1.1`, and `TLSv1.2`. By default, the `SSLv3` and `TLSv1.2` protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
HTTPSSSLProtocol TLSv1.2
```

SNI

This optional directive specifies the host name used for Server Name Indication (SNI) in HTTPS mode.

89.8.2. Procedures

The following procedures are exported by om_http.

```
add_http_header(string name, string value);
```

Dynamically add a custom HTTP header to HTTP requests.

```
set_http_request_path(string path);
```

Set the *path* in the HTTP request to the string specified. This is useful if the URL is dynamic and parameters such as event ID need to be included in the URL. Note that the string must be URL encoded if it contains reserved characters.

89.8.3. Examples

Example 495. Sending Logs over HTTPS

This configuration reads log messages from file and forwards them via HTTPS.

```
nxlog.conf
1 <Input file>
2   Module      im_file
3   File        'input.log'
4 </Input>
5
6 <Output http>
7   Module      om_http
8   URL         https://server:8080/
9   HTTPSCertFile    %CERTDIR%/client-cert.pem
10  HTTPSCertKeyFile  %CERTDIR%/client-key.pem
11  HTTPSCAFile      %CERTDIR%/ca.pem
12  HTTPSAllowUntrusted FALSE
13 </Output>
14
15 <Route file_to_http>
16   Path        file => http
17 </Route>
```

89.9. Kafka (om_kafka)

This module implements an [Apache Kafka](#) producer for publishing event records to a Kafka topic. See also the [im_kafka](#) module.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.9.1. Configuration

The `om_kafka` module accepts the following directives in addition to the [common module directives](#). The **BrokerList** and **Topic** directives are required.

BrokerList

This mandatory directive specifies the list of Kafka brokers to connect to for publishing logs. The list should include ports and be comma-delimited (for example, `localhost:9092, 192.168.88.35:19092`).

Topic

This mandatory directive specifies the Kafka topic to publish records to.

CAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote brokers. **CAFile** is required if **Protocol** is set to `ssl`.

CertFile

This specifies the path of the certificate file to be used for the SSL handshake.

CertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake.

Compression

This directive specifies the compression types to use during transfer. Available types depend on the Kafka library, and should include `none` (the default), `gzip`, `snappy`, and `lz4`.

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [**CertKeyFile**](#). This directive is not needed for passwordless private keys.

Partition

This optional integer directive specifies the topic partition to write to. If this directive is not given, messages are sent without a partition specified.

Protocol

This optional directive specifies the protocol to use for connecting to the Kafka brokers. Accepted values include `plaintext` (the default) and `ssl`. If **Protocol** is set to `ssl`, then the **CAFile** directive must also be provided.

89.9.2. Examples

Example 496. Using the `om_kafka` Module

This configuration sends events to a Kafka cluster using the brokers specified. Events are published to the first partition of the `nxlog` topic.

`nxlog.conf`

```
1 <Output out>
2   Module      om_kafka
3   BrokerList  localhost:9092,192.168.88.35:19092
4   Topic       nxlog
5   Partition   0
6   Protocol    ssl
7   CAFile      /root/ssl/ca-cert
8   CertFile    /root/ssl/client_debian-8.pem
9   CertKeyFile /root/ssl/client_debian-8.key
10  KeyPass     thisisasecret
11 </Output>
```

89.10. Null (om_null)

Log messages sent to the `om_null` module instance are discarded, this module does not write its output anywhere. It can be useful for creating a dummy route, for testing purposes, or for [Scheduled NXLog code execution](#). The `om_null` module accepts only the [common module directives](#). See [this example](#) for usage.

89.11. Oracle (om_oci)

This module can write log messages to an Oracle database.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.11.1. Configuration

The `om_oci` module accepts the following directives in addition to the [common module directives](#). The `DBname`, `Password`, and `UserName` directives are required.

DBname

Name of the database to write the logs to.

Password

Password for authenticating to the database server.

UserName

Username for authenticating to the database server.

ORACLE_HOME

This optional directive specifies the directory of the Oracle installation.

SQL

An optional SQL statement to override the default.

89.11.2. Examples

Example 497. Storing Logs in an Oracle Database

This configuration reads BSD Syslog messages from socket, parses the messages, and inserts them into the database.

nxlog.conf

```
1 <Extension _syslog>
2     Module      xm_syslog
3 </Extension>
4
5 <Input uds>
6     Module      im_uds
7     UDS        /dev/log
8 </Input>
9
10 <Output oci>
11    Module      om_oci
12    dbname     //192.168.1.1:1521/orcl
13    username   joe
14    password   secret
15    SQL        INSERT INTO log ("id", "facility", "severity", "hostname", \
16                                "timestamp", "application", "message") \
17                                VALUES (log_seq.nextval, $SyslogFacility, $SyslogSeverity, \
18                                $Hostname, to_date($rcvd_timestamp, \
19                                'YYYY-MM-DD HH24:MI:SS'), \
20                                $SourceName, $Message)
21    Exec       parse_syslog();
22 </Output>
23
24 <Route uds_to_oci>
25     Path      uds => oci
26 </Route>
```

89.12. ODBC (om_odbc)

ODBC is a database independent abstraction layer for accessing databases. This module uses the ODBC API to write data to database tables. There are several ODBC implementations available, and this module has been tested with unixODBC on Linux (available in most major distributions) and Microsoft ODBC on Windows.

Setting up the ODBC data source is not in the scope of this document. Please consult the relevant ODBC guide: the [unixODBC documentation](#) or the [Microsoft ODBC Data Source Administrator guide](#). The data source must be accessible by the user NXLog is running under.

NOTE

The "SQL Server" ODBC driver is unsupported and does not work. Please use the "SQL Server Native Client" ODBC driver to insert records into a Microsoft SQL Server database.

Unlike other database modules, this module has no directive to specify the SQL statement which is executed. The module provides two functions, [sql_exec\(\)](#) and [sql_fetch\(\)](#), which can be executed using the [Exec](#) directive. This allows more complex processing rules to be used and also makes it possible to insert records into more than one table.

NOTE

Both `sql_exec()` and `sql_fetch()` can take bind parameters as function arguments. It is recommended to use bind parameters instead of concatenating the SQL statement with the value. For example, these two are equivalent but the first is dangerous due to the lack of escaping:

```
$retval = sql_exec("INSERT INTO log (id) VALUES (" + $id + ")");
```

```
$retval = sql_exec("INSERT INTO log (id) VALUES (?)", $id);
```

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.12.1. Configuration

The `om_odbc` module accepts the following directives in addition to the [common module directives](#).

ConnectionString

The mandatory directive specifies the ODBC data source connection string.

89.12.2. Functions

The following functions are exported by `om_odbc`.

`string sql_error()`

Return the error message from the last failed ODBC operation.

`boolean sql_exec(string statement, varargs args)`

Execute the SQL *statement*. Bind parameters should be passed separately after the *statement* string. Returns FALSE if the SQL execution failed: `sql_error()` can be used to retrieve the error message.

`boolean sql_fetch(string statement, varargs args)`

Fetch the first row of the result set specified with a SELECT query in *statement*. The function will create or populate fields named after the columns in the result set. Bind parameters should be passed separately after the *statement* string. Returns FALSE if the SQL execution failed: `sql_error()` can be used to retrieve the error message.

89.12.3. Examples

Example 498. Complex Write to an ODBC Data Source

In this example, the events read from the TCP input are inserted into the *message* column. The table has an auto_increment *id* column, which is used to fetch and print the newly inserted line.

nxlog.conf

```

1 <Input tcp>
2   Module      im_tcp
3   Port        1234
4   Host        0.0.0.0
5 </Input>
6
7 <Output odbc>
8   Module      om_odbc
9   ConnectionString DSN=mysql_ds;username=mysql;password=mysql;database=logdb;
10  <Exec>
11    if ( sql_exec("INSERT INTO log (facility, severity, hostname, timestamp, " +
12          "application, message) VALUES (?, ?, ?, ?, ?, ?)",
13          1, 2, "host", now(), "app", $raw_event) == TRUE )
14    {
15      if ( sql_fetch("SELECT max(id) as id from log") == TRUE )
16      {
17        log_info("ID: " + $id);
18        if ( sql_fetch("SELECT message from log WHERE id=?", $id) == TRUE )
19        {
20          log_info($message);
21        }
22      }
23    }
24  </Exec>
25 </Output>
26
27 <Route tcp_to_odbc>
28   Path        tcp => odbc
29 </Route>
```

89.13. Perl (om_perl)

The [Perl programming language](#) is widely used for log processing and comes with a broad set of modules bundled or available from [CPAN](#). Code can be written more quickly in Perl than in C, and code execution is safer because exceptions (`croak/die`) are handled properly and will only result in an unfinished attempt at log processing rather than taking down the whole NXLog process.

This module makes it possible to execute Perl code in an output module that can handle the data directly in Perl. See also the [im_perl](#) and [xm_perl](#) modules.

The module will parse the file specified in the [PerlCode](#) directive when NXLog starts the module. The Perl code must implement the *write_data* subroutine which will be called by the module when there is data to process. This subroutine is called for each event record and the event record is passed as an argument. To access event data, the Log::Nxlog Perl module must be included, which provides the following methods.

log_debug(msg)

Send the message *msg* to the internal logger on DEBUG log level. This method does the same as the [log_debug\(\)](#) procedure in NXLog.

log_info(msg)

Send the message *msg* to the internal logger on INFO log level. This method does the same as the [log_info\(\)](#)

procedure in NXLog.

`log_warning(msg)`

Send the message `msg` to the internal logger on WARNING log level. This method does the same as the [log_warning\(\)](#) procedure in NXLog.

`log_error(msg)`

Send the message `msg` to the internal logger on ERROR log level. This method does the same as the [log_error\(\)](#) procedure in NXLog.

`get_field(event, key)`

Retrieve the value associated with the field named `key`. The method returns a scalar value if the key exists and the value is defined, otherwise it returns `undef`.

For the full NXLog Perl API, see the POD documentation in [Nxlog.pm](#). The documentation can be read with `perldoc Log::Nxlog`.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.13.1. Configuration

The `om_perl` module accepts the following directives in addition to the [common module directives](#).

PerlCode

This mandatory directive expects a file containing valid Perl code. This file is read and parsed by the Perl interpreter.

89.13.2. Examples

Example 499. Handling Event Data in `om_perl`

This output module sends events to the Perl script, which simply writes the data from the `$raw_event` field into a file.

`nxlog.conf`

```
1 <Output out>
2   Module      om_perl
3   PerlCode   modules/output/perl/perl-output.pl
4 </Output>
```

`perl-output.pl`

```
use strict;
use warnings;

use Log::Nxlog;

sub write_data
{
    my ($event) = @_;
    my $rawevt = Log::Nxlog::get_field($event, 'raw_event');
    open(OUT, '>', 'tmp/output') || die("cannot open tmp/output: $!");
    print OUT $rawevt, "(from perl)", "\n";
    close(OUT);
}
```

89.14. Python (om_python)

This module provides support for forwarding log data with methods written in the [Python](#) language. The file specified by the `PythonCode` directive should contain a `write_data()` method which is called by the `om_python` module instance. See also the [xm_python](#) and [im_python](#) modules.

The Python script should import the `nxlog` module, and will have access to the following classes and functions.

`nxlog.log_debug(msg)`

Send the message `msg` to the internal logger at DEBUG log level. This function does the same as the core `log_debug()` procedure.

`nxlog.log_info(msg)`

Send the message `msg` to the internal logger at INFO log level. This function does the same as the core `log_info()` procedure.

`nxlog.log_warning(msg)`

Send the message `msg` to the internal logger at WARNING log level. This function does the same as the core `log_warning()` procedure.

`nxlog.log_error(msg)`

Send the message `msg` to the internal logger at ERROR log level. This function does the same as the core `log_error()` procedure.

`class nxlog.Module()`

This class is instantiated by NXLog and can be accessed via the `LogData().module` attribute. This can be used to set or access variables associated with the module (see the [example](#) below).

`class nxlog.LogData()`

This class represents an event. It is instantiated by NXLog and passed to the `write_data()` method.

`delete_field(name)`

This method removes the field `name` from the event record.

`field_names()`

This method returns a list with the names of all the fields currently in the event record.

`get_field(name)`

This method returns the value of the field `name` in the event.

`set_field(name, value)`

This method sets the value of field `name` to `value`.

`module`

This attribute is set to the `Module` object associated with the `LogData` event.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.14.1. Configuration

The `om_python` module accepts the following directives in addition to the [common module directives](#).

`PythonCode`

This mandatory directive specifies a file containing Python code. The `om_python` instance will call a `write_data()` function which must accept an `nxlog.LogData()` object as its only argument.

89.14.2. Examples

Example 500. Forwarding Events With om_python

This example shows an alerter implemented as an output module instance in Python. First, any event with a normalized severity less than of 4/ERROR is dropped; see the `Exec` directive (`xm_syslog` and most other modules set a normalized `$SeverityValue` field). Then the Python function generates a custom email and sends it via SMTP.

`nxlog.conf`

```
1 <Output out>
2   Module      om_python
3   PythonCode  /opt/nxlog/etc/output.py
4   Exec        if $SeverityValue < 4 drop();
5 </Output>
```

`output.py`

```
from email.mime.text import MIMEText
import pprint
import smtplib
import socket

import nxlog

HOSTNAME = socket.gethostname()
FROM_ADDR = 'nxlog@{}'.format(HOSTNAME)
TO_ADDR = 'you@example.com'

def write_data(event):
    nxlog.log_debug('Python alerter received event')

    # Convert field list to dictionary
    all = {}
    for field in event.get_names():
        all.update({field: event.get_field(field)})

    # Create message from event
    pretty_event = pprint.pformat(all)
    msg = 'NXLog on {} received the following alert:\n\n{}'.format(HOSTNAME,
        pretty_event)
    email_msg = MIMEText(msg)
    email_msg['Subject'] = 'Alert from NXLog on {}'.format(HOSTNAME)
    email_msg['From'] = FROM_ADDR
    email_msg['To'] = TO_ADDR

    # Send email message
    nxlog.log_debug('Sending email alert for event')
    s = smtplib.SMTP('localhost')
    s.sendmail(FROM_ADDR, [TO_ADDR], email_msg.as_string())
    s.quit()
```

89.15. Redis (om_redis)

This module can store data in a Redis server. It issues `RPUSH` commands using the [Redis Protocol](#) to send data.

The input counterpart, `im_redis`, can be used to retrieve data from a Redis server.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User

Guide.

89.15.1. Configuration

The *om_redis* module accepts the following directives in addition to the [common module directives](#). The [Host](#) directive is required.

Host

This mandatory directive specifies the IP address or DNS hostname of the Redis server to connect to.

Command

This optional directive specifies the command to be used. The possible commands are [LPUSH](#), [RPUSH](#) (the default), [LPUSHX](#), and [RPUSHX](#).

Key

This specifies the *Key* used by the [RPUSH](#) command. It must be a [string](#) type expression. If the expression in the **Key** directive is not a constant string (it contains functions, field names, or operators), it will be evaluated for each event to be inserted. The default is [nxlog](#).

OutputType

See the [OutputType](#) directive in the list of common module directives. If this directive is unset, the default [Dgram](#) formatter function is used, which writes the value of [\\$raw_event](#) without a line terminator. To preserve structured data [Binary](#) can be used, but it must also be set on the other end.

Port

This specifies the port number of the Redis server. The default is port 6379.

89.16. Ruby (om_ruby)

This module provides support for forwarding log data with methods written in the [Ruby](#) language. See also the [xm_ruby](#) and [im_ruby](#) modules.

The [Nxlog](#) module provides the following classes and methods.

Nxlog.log_info(msg)

Send the message *msg* to the internal logger at DEBUG log level. This method does the same as the core [log_debug\(\)](#) procedure.

Nxlog.log_debug(msg)

Send the message *msg* to the internal logger at INFO log level. This method does the same as the core [log_info\(\)](#) procedure.

Nxlog.log_warning(msg)

Send the message *msg* to the internal logger at WARNING log level. This method does the same as the core [log_warning\(\)](#) procedure.

Nxlog.log_error(msg)

Send the message *msg* to the internal logger at ERROR log level. This method does the same as the core [log_error\(\)](#) procedure.

class Nxlog.Module

This class will be instantiated by NXLog and passed to the method specified by the [Call](#) directive.

`class Nxlog.LogData`

This class represents an event. It is instantiated by NXLog and passed to the method specified by the [Call](#) directive.

`field_names()`

This method returns an array with the names of all the fields currently in the event record.

`get_field(name)`

This method returns the value of the field *name* in the event.

`set_field(name, value)`

This method sets the value of field *name* to *value*.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.16.1. Configuration

The *om_ruby* module accepts the following directives in addition to the [common module directives](#). The [RubyCode](#) directive is required.

`RubyCode`

This mandatory directive specifies a file containing Ruby code. The *om_ruby* instance will call the method specified by the [Call](#) directive. The method must accept [Nxlog.Module](#) and [Nxlog.LogData](#) objects as arguments.

`Call`

This optional directive specifies the Ruby method to call. The default is [write_data](#).

89.16.2. Examples

Example 501. Forwarding Events With `om_ruby`

This example uses a Ruby script to choose an output file according to the severity of the event. Normalized severity fields are added by most modules; see, for example, the `xm_syslog $SeverityValue` field.

TIP

See [Using Dynamic Filenames](#) for a way to implement this functionality natively.

`nxlog.conf`

```
1 <Output out>
2   Module      om_ruby
3   RubyCode   ./modules/output/ruby/proc2.rb
4   Call        write_data
5 </Output>
```

`om_ruby2.rb`

```
def write_data(mod, event)
  if event.get_field('SeverityValue') >= 4
    Nxlog.log_debug('Writing out high severity event')
    File.open('tmp/high_severity', 'a') do |file|
      file.write("#{event.get_field('raw_event')}\n")
      file.flush
    end
  else
    Nxlog.log_debug('Writing out low severity event')
    File.open('tmp/low_severity', 'a') do |file|
      file.write("#{event.get_field('raw_event')}\n")
      file.flush
    end
  end
end
```

89.17. TLS/SSL (`om_ssl`)

The `om_ssl` module uses the OpenSSL library to provide an SSL/TLS transport. It behaves like the [om_tcp](#) module, except that an SSL handshake is performed at connection time and the data is received over a secure channel. Log messages transferred over plain TCP can be eavesdropped or even altered with a man-in-the-middle attack, while the `om_ssl` module provides a secure log message transport.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.17.1. Configuration

The `om_ssl` module accepts the following directives in addition to the [common module directives](#). The `Host` directive is required.

`Host`

The module will connect to this IP address or DNS hostname.

`Port`

The module will connect to this port number on the remote host. The default is port 514.

`AllowUntrusted`

This boolean directive specifies that the connection should be allowed without certificate verification. If set to

TRUE the connection will be allowed even if the remote server presents an unknown or self-signed certificate. The default value is FALSE: the remote socket must present a trusted certificate.

CADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format.

CAFfile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote socket.

CertFile

This specifies the path of the certificate file to be used for the SSL handshake.

CertKeyFile

This specifies the path of the certificate key file to be used for the SSL handshake.

CRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote socket. The certificate filenames in this directory must be in the OpenSSL hashed format.

CRLFile

This specifies the path of the certificate revocation list (CRL) which will be used to check the certificate of the remote socket against.

KeyPass

With this directive, a password can be supplied for the certificate key file defined in [CertKeyFile](#). This directive is not needed for passwordless private keys.

LocalPort

This optional directive specifies the local port number of the connection. If this is not specified a random high port number will be used, which is not always ideal in firewalled network environments.

OutputType

See the [OutputType](#) directive in the list of common module directives. The default is [LineBased_LF](#).

Reconnect

This directive has been deprecated as of version 2.4. The module will try to reconnect automatically at increasing intervals on all errors.

SNI

This optional directive specifies the host name used for Server Name Indication (SNI).

SSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

SSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
SSLProtocol TLSv1.2
```

89.17.2. Procedures

The following procedures are exported by `om_ssl`.

reconnect();

Force a reconnection. This can be used from a Schedule block to periodically reconnect to the server.

89.17.3. Examples

Example 502. Sending Binary Data to Another NXLog Agent

This configuration reads log messages from socket and sends them in the NXLog **binary** format to another NXLog agent.

`nxlog.conf`

```
1 <Input uds>
2   Module      im_uds
3   UDS        tmp/socket
4 </Input>
5
6 <Output ssl>
7   Module      om_ssl
8   Host        localhost
9   Port        23456
10  CAFile     %CERTDIR%/ca.pem
11  CertFile   %CERTDIR%/client-cert.pem
12  CertKeyFile %CERTDIR%/client-key.pem
13  KeyPass    secret
14  AllowUntrusted TRUE
15  OutputType Binary
16 </Output>
17
18 <Route uds_to_ssl>
19   Path        uds => ssl
20 </Route>
```

89.18. TCP (om_tcp)

This module initiates a TCP connection to a remote host and transfers log messages. Or, in **Listen** mode, this module accepts client connections and multiplexes data to all connected clients. The TCP transfer protocol provides more reliable log transmission than UDP. If security is a concern, consider using the `om_ssl` module instead.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.18.1. Configuration

The `om_tcp` module accepts the following directives in addition to the [common module directives](#). The `Host` directive is required.

Host

The module will connect to this IP address or DNS hostname. Or, if `Listen` is set to TRUE, the module will listen for connections on this address.

Port

The module will connect to this port number on the remote host. Or, if `Listen` is set to TRUE, the module will listen for connections on this port. The default is port 514.

Listen

If TRUE, this boolean directive specifies that `om_tcp` should listen for connections at the local address specified by the `Host` directive rather than opening a connection to the address. The default is FALSE: `om_tcp` will connect to the specified address.

LocalPort

This optional directive specifies the local port number of the connection. This directive only applies if `Listen` is set to TRUE. If this is not specified a random high port number will be used, which is not always ideal in firewalled network environments.

OutputType

See the [OutputType](#) directive in the list of common module directives. The default is [LineBased_LF](#).

QueueInListenMode

If set to TRUE, this boolean directive specifies that events should be queued if no client is connected. If this module's buffer becomes full, the preceding module in the route will be paused or events will be dropped, depending on whether `FlowControl` is enabled. This directive only applies if `Listen` is set to TRUE. The default is FALSE: `om_tcp` will discard events if no client is connected.

Reconnect

This directive has been deprecated as of version 2.4. The module will try to reconnect automatically at increasing intervals on all errors.

89.18.2. Procedures

The following procedures are exported by `om_tcp`.

`reconnect();`

Force a reconnection. This can be used from a Schedule block to periodically reconnect to the server.

89.18.3. Examples

Example 503. Transferring Raw Logs over TCP

With this configuration, NXLog will read log messages from socket and forward them via TCP.

nxlog.conf

```
1 <Input uds>
2   Module im_uds
3   UDS    /dev/log
4 </Input>
5
6 <Output tcp>
7   Module om_tcp
8   Host   192.168.1.1
9   Port   1514
10 </Output>
11
12 <Route uds_to_tcp>
13   Path   uds => tcp
14 </Route>
```

89.19. UDP (om_udp)

This module sends log messages as UDP datagrams to the address and port specified. UDP is the transport protocol of the legacy BSD Syslog standard as described in RFC 3164, so this module can be particularly useful to send messages to devices or Syslog daemons which do not support other transports.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.19.1. Configuration

The *om_udp* module accepts the following directives in addition to the [common module directives](#). The **Host** directive is required.

Host

The module will connect to this IP address or DNS hostname.

Port

The module will connect to this port number on the remote host. The default is port 514.

LocalPort

This optional directive specifies the local port number of the connection. If this is not specified a random high port number will be used, which is not always ideal in firewalled network environments.

SockBufSize

This optional directive sets the socket buffer size (SO_SNDBUF) to the value specified. If this is not set, the operating system default is used.

89.19.2. Examples

Example 504. Sending Raw Syslog over UDP

This configuration reads log messages from socket and forwards them via UDP.

nxlog.conf

```

1 <Input uds>
2   Module im_uds
3   UDS    /dev/log
4 </Input>
5
6 <Output udp>
7   Module om_udp
8   Host   192.168.1.1
9   Port   1514
10 </Output>
11
12 <Route uds_to_udp>
13   Path   uds => udp
14 </Route>
```

89.20. UDP with IP Spoofing (om_udpspoof)

This module sends log messages as UDP datagrams to the address and port specified and allows the source address in the UDP packet to be spoofed in order to make the packets appear as if they were sent from another host. This is particularly useful in situations where log data needs to be forwarded to another server and the server uses the client address to identify the data source. With IP spoofing the UDP packets will contain the IP address of the originating client that produced the message instead of the forwarding server.

This module is very similar to the [om_udp](#) module and can be used as a drop-in replacement. The [SpoofAddress](#) configuration directive can be used to set the address if necessary. The UDP datagram will be sent with the local IP address if the IP address to be spoofed is invalid. The source port in the UDP datagram will be set to the port number of the local connection (the port number is not spoofed).

The network input modules ([im_udp](#), [im_tcp](#), and [im_ssl](#)) all set the [\\$MessageSourceAddress](#) field, and this value will be used when sending the UDP datagrams (unless [SpoofAddress](#) is explicitly set to something else). This allows logs to be collected over reliable and secure transports (like SSL), while the [om_udpspoof](#) module is only used for forwarding to the destination server that requires spoofed UDP input.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.20.1. Configuration

The [om_udpspoof](#) module accepts the following directives in addition to the [common module directives](#). The [Host](#) directive is required.

Host

The module will send UDP datagrams to this IP address or DNS hostname.

Port

The module will send UDP packets to this port. The default port is 514 if this directive is not specified.

LocalPort

This optional directive specifies the local port number of the connection. If this is not specified a random high port number will be used which is not always ideal in firewalled network environments.

MTU

This directive can be used to specify the maximum transfer size of the IP data fragments. If this value exceeds the MTU size of the sending interface, an error may occur and the packet be dropped. The default MTU value is **1500**.

SockBufSize

This optional directive sets the socket buffer size (SO_SNDBUF) to the value specified. If this is not set, the operating system default is used.

SpooftAddress

This directive is optional. The IP address rewrite takes place depending on how this directive is specified.

Directive not specified

The IP address stored in the **\$MessageSourceAddress** field is used so the module should work automatically when the **SpooftAddress** directive is not specified.

Constant literal value

The literal value may be a **string** or an **ip4addr** type. For example, **SpooftAddress '10.0.0.42'** and **SpooftAddress 10.0.0.42** are equivalent.

Expression

The expression specified here will be evaluated for each message to be sent. Normally this can be a field name, but anything is accepted which evaluates to a **string** or an **ip4addr** type. For example, **SpooftAddress \$MessageSourceAddress** has the same effect as when **SpooftAddress** is not set.

89.20.2. Examples

Example 505. Simple Forwarding with IP Address Spoofing

The **im_tcp** module will accept log messages via TCP and will set the **\$MessageSourceAddress** field for each event. This value will be used by **om_udpspoof** to set the UDP source address when sending the data to **logserver** via UDP.

nxlog.conf

```
1 <Input tcp>
2   Module  im_tcp
3   Host    0.0.0.0
4   Port    1514
5 </Input>
6
7 <Output udpspoof>
8   Module  om_udpspoof
9   Host    logserver.example.com
10  Port    1514
11 </Output>
12
13 <Route tcp_to_udpspoof>
14   Path    tcp => udpspoof
15 </Route>
```

Example 506. Forwarding Log Messages with Spoofed IP Address from Multiple Sources

This configuration accepts log messages via TCP and UDP, and also reads messages from a file. Both `im_tcp` and `im_udp` set the `$MessageSourceAddress` field for incoming messages, and in both cases this is used to set `$sourceaddr`. The `im_file` module instance is configured to set the `$sourceaddr` field to `10.1.2.3` for all log messages. Finally, the `om_udpspoof` output module instance is configured to read the value of the `$sourceaddr` field for spoofing the UDP source address.

`nxlog.conf`

```

1 <Input tcp>
2   Module      im_tcp
3   Host        0.0.0.0
4   Port        1514
5   Exec        $sourceaddr = $MessageSourceAddress;
6 </Input>
7
8 <Input udp>
9   Module      im_udp
10  Host        0.0.0.0
11  Port        1514
12  Exec        $sourceaddr = $MessageSourceAddress;
13 </Input>
14
15 <Input file>
16  Module      im_file
17  File        '/var/log/myapp.log'
18  Exec        $sourceaddr = 10.1.2.3;
19 </Input>
20
21 <Output udpspoof>
22  Module      om_udpspoof
23  SpoofAddress $sourceaddr
24  Host        10.0.0.1
25  Port        1514
26 </Output>
27
28 <Route all_to_file>
29  Path        tcp, udp, file => udpspoof
30 </Route>
```

89.21. UDS (om_uds)

This module allows log messages to be sent to a Unix domain socket. Unix systems traditionally have a `/dev/log` or similar socket used by the system logger to accept messages. Applications use the `syslog(3)` system call to send messages to the system logger. NXLog can use this module to send log messages to another Syslog daemon via the socket.

NOTE

This module supports `SOCK_DGRAM` type sockets only. `SOCK_STREAM` type sockets may be supported in the future.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.21.1. Configuration

The `om_uds` module accepts the following directives in addition to the [common module directives](#).

UDS

This specifies the path of the Unix domain socket. The default is `/dev/log`.

UDSType

This directive specifies the domain socket type. Supported values are `dgram`, `stream`, and `auto`. The default is `auto`.

OutputType

See the `OutputType` directive in the list of common module directives. If `UDSType` is set to `Dgram` or is set to `auto` and a SOCK_DGRAM type socket is detected, this defaults to `Dgram`. If `UDSType` is set to `stream` or is set to `auto` and a SOCK_STREAM type socket is detected, this defaults to `LineBased`.

89.21.2. Examples

Example 507. Using the `om_uds` Module

This configuration reads log messages from a file, adds BSD Syslog headers with default fields, and writes the messages to socket.

`nxlog.conf`

```
1 <Extension syslog>
2   Module xm_syslog
3 </Extension>
4
5 <Input file>
6   Module im_file
7   File   "/var/log/custom_app.log"
8 </Input>
9
10 <Output uds>
11   Module om_uds
12   # Defaulting Syslog fields and creating Syslog output
13   Exec   parse_syslog_bsd(); to_syslog_bsd();
14   UDS    /dev/log
15 </Output>
16
17 <Route file_to_uds>
18   Path   file => uds
19 </Route>
```

89.22. WebHDFS (om_webhdfs)

This module allows logs to be stored in Hadoop HDFS using the WebHDFS protocol.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.22.1. Configuration

The `om_webhdfs` module accepts the following directives in addition to the [common module directives](#). The `File` and `URL` directives are required.

File

This mandatory directive specifies the name of the destination file. It must be a [string type expression](#). If the expression in the **File** directive is not a constant string (it contains functions, field names, or operators), it will be evaluated before each request is dispatched to the WebHDFS REST endpoint (and after the **Exec** is evaluated). Note that the filename must be quoted to be a valid string literal, unlike in other directives which take a filename argument.

URL

This mandatory directive specifies the URL of the WebHDFS REST endpoint where the module should POST the event data. The module operates in plain HTTP or HTTPS mode depending on the URL provided, and connects to the hostname specified in the URL. If the port number is not explicitly indicated in the URL, it defaults to port 80 for HTTP and port 443 for HTTPS.

FlushInterval

The module will send the data to the endpoint defined in [URL](#) after this amount of time in seconds, unless [FlushLimit](#) is reached first. This defaults to 5 seconds.

FlushLimit

When the number of events in the output buffer reaches the value specified by this directive, the module will send the data to the endpoint defined in [URL](#). This defaults to 500 events. The [FlushInterval](#) may trigger sending the write request before this limit is reached if the log volume is low to ensure that data is sent promptly.

HTTPSAAllowUntrusted

This boolean directive specifies that the connection should be allowed without certificate verification. If set to TRUE, the connection will be allowed even if the remote HTTPS server presents an unknown or self-signed certificate. The default value is FALSE: the remote must present a trusted certificate.

HTTPSCADir

This specifies the path to a directory containing certificate authority (CA) certificates, which will be used to check the certificate of the remote HTTPS server. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCAFile

This specifies the path of the certificate authority (CA) certificate, which will be used to check the certificate of the remote HTTPS server.

HTTPSCertFile

This specifies the path of the certificate file to be used for the HTTPS handshake.

HTTPSCertKeyFile

This specifies the path of the certificate key file to be used for the HTTPS handshake.

HTTPSCRLDir

This specifies the path to a directory containing certificate revocation lists (CRLs), which will be consulted when checking the certificate of the remote HTTPS server. The certificate filenames in this directory must be in the OpenSSL hashed format.

HTTPSCRLFile

This specifies the path of the certificate revocation list (CRL), which will be consulted when checking the certificate of the remote HTTPS server.

HTTPSKeyPass

With this directive, a password can be supplied for the certificate key file defined in [HTTPSCertKeyFile](#). This directive is not needed for passwordless private keys.

HTTPSSLCipher

This optional directive can be used to set the permitted SSL cipher list, overriding the default. Use the format described in the [ciphers\(1ssl\)](#) man page.

HTTPSSLProtocol

This directive can be used to set the allowed SSL/TLS protocol(s). It takes a comma-separated list of values which can be any of the following: **SSLv2**, **SSLv3**, **TLSv1**, **TLSv1.1**, and **TLSv1.2**. By default, the SSLv3 and TLSv1.2 protocols are allowed.

WARNING

Due to the various vulnerabilities discovered in the SSL/TLS protocols, only TLSv1.2 is now considered secure. Earlier NXLog versions (2.8.1296 and before) supported SSLv3 only, and not TLSv1.2. If **SSLProtocol** is not specified, the vulnerable SSLv3 protocol will be allowed for compatibility with older versions. It is recommended to explicitly allow TLSv1.2 only by defining the following in the configuration:

```
HTTPSSLProtocol TLSv1.2
```

QueryParam

This configuration option can be used to specify additional HTTP Query Parameters such as *BlockSize*. This option may be used to define more than one parameter:

```
QueryParam blocksize 42
QueryParam destination /foo
```

89.22.2. Examples

Example 508. Sending Logs to a WebHDFS Server

This example output module instance forwards messages to the specified URL and file using the WebHDFS protocol.

nxlog.conf

```
1 <Output hdf5>
2   Module      om_webhdfs
3   URL        http://hdfsserver.domain.com/
4   File        "myfile"
5   QueryParam  blocksize 42
6   QueryParam  destination /foo
7 </Output>
```

89.23. ZeroMQ (om_zmq)

This module provides message transport over [ZeroMQ](#), a scalable high-throughput messaging library.

See the [im_zmq](#) for the input pair of this module.

See the [list of installer packages](#) that provide this module in the Available Modules chapter of the NXLog User Guide.

89.23.1. Configuration

The *om_zmq* module accepts the following directives in addition to the [common module directives](#). The [Address](#), [ConnectionType](#), [Port](#), and [SocketType](#) directives are required.

Address

This directive specifies the ZeroMQ socket address.

ConnectionType

This mandatory directive specifies the underlying transport protocol. It may be one of the following: [TCP](#), [PGM](#), or [EPGM](#).

Port

This directive specifies the ZeroMQ socket port.

SocketType

This mandatory directive defines the type of the socket to be used. It may be one of the following: [REP](#), [ROUTER](#), [PUB](#), [XPUB](#), or [PUSH](#). This must be set to [SUB](#) if [ConnectionType](#) is set to [PGM](#) or [EPGM](#).

Interface

This directive specifies the ZeroMQ socket interface.

OutputType

See the [OutputType](#) directive in the list of common module directives. The default value is [Dgram](#).

89.23.2. Examples

Example 509. Using the om_zmq Module

This example configuration reads log messages from file and forwards them via ZeroMQ PUSH socket over TCP.

nxlog.conf

```
1 <Input file>
2   Module      im_file
3   File        "/var/log/messages"
4 </Input>
5
6 <Output zmq>
7   Module      om_zmq
8   SocketType  PUSH
9   ConnectionType  TCP
10  Address     10.0.0.1
11  Port        1514
12 </Output>
13
14 <Route file_to_zmq>
15   Path        file => zmq
16 </Route>
```

NXLog Manager

Chapter 90. Introduction

Managing a log collection system where agents are scattered around the whole network can be a daunting task especially if there are multiple teams in charge of each system.

The NXLog Manager is a log management solution which provides a web based administration interface to configure all parameters for the log collection and enables the log management administrator to efficiently monitor and manage the NXLog agents securely from a central console. The NXLog Manager is able to operate in clustered mode if the network topology requires multiple manager nodes.

This document provides information about the following topics:

- Installation steps for the core NXLog Manager system.
- Installation steps for the NXLog agents to be deployed on the client machines.
- Details about each component of the NXLog Manager system accessible from the web interface.

90.1. Requirements

To use and administer NXLog Manager, the user is expected to be familiar with the following:

- Using Mozilla Firefox or compatible web browser.
- Regular expressions.
- Concept of X509 certificates and public key cryptography.
- Log management basics.
- Networking concepts.

The web interface supports the following browsers:

- Mozilla Firefox 3.5 or higher.
- Google Chrome 10 or higher.

There are known problems with Microsoft Internet Explorer and it is not supported.

90.2. Architecture

NXLog Manager web application

NXLog Manager is a java based web application which can communicate with the NXLog agents.

NXLog

NXLog is the log collector with no frontend. NXLog can be used in both server and client mode. When running as a client (agent), NXLog will collect local log sources and will forward the data over the network. NXLog can also operate as a server to store messages locally or as a relay to forward messages to another instance.

The architecture of NXLog Manager allows log collection to function even if NXLog Manager is not running or the control channel is not functional, thus an NXLog Manager upgrade will not cause any interruption to the log collection process.

Chapter 91. Installation

91.1. Installing on Debian Wheezy

Assuming we have the deb packages in the working directory these can be installed with the following commands:

```
# dpkg -i nxlog-manager_X.X.XXX_amd64.deb  
# apt-get -f install
```

NOTE

nxlog-manager-4.X.XXXX.deb requires openjdk-7-jre and nxlog-manager-5.X.XXXX.deb requires either openjdk-7-jre or openjdk-8-jre. If java is not installed or the correct version of java is not selected NXLog Manager will refuse to start. To select the default version of java on your system, use the command:

```
# update-alternatives --config java
```

CAUTION

Make sure that your hostname and DNS settings are setup correctly, to avoid problems with NXLog Manager. Refer to [host setup common issues](#) for more information.

91.2. Installing on Redhat 6 & 7

With rpm packages in the local directory type the following to install the packages.

```
# yum localinstall nxlog-manager-X.X.XXXX-1.noarch.rpm
```

NOTE

nxlog-manager-4.X.XXXX-1.noarch.rpm requires java-1.7.0-openjdk and nxlog-manager-5.X.XXXX-1.noarch.rpm requires either java-1.7.0-openjdk or java-1.8.0-openjdk. If java is not installed or the correct version of java is not selected NXLog Manager will refuse to start. To select the default version of java on your system, use the command:

```
# alternatives --config java
```

If you want to access the web interface from another host, please make sure that your firewall rules allow access to port 9090 from the external network:

```
# iptables -A INPUT -p tcp --dport 9090 -j ACCEPT
```

Or completely remove all firewall rules while testing:

```
# iptables -F
```

CAUTION

Make sure that your hostname and DNS settings are setup correctly, to avoid problems with NXLog Manager. Refer to [host setup common issues](#) for more information.

91.3. Installing as Docker application

To install NXLog Manager as a Docker application the Docker Engine as well as the Docker Compose tool is required. The procedure is identical on all platforms supported by Docker (Linux, Windows and MacOS). Extract the files from the compressed Docker archive.

```
$ tar zxf nxlog-manager-X.X.XXXX-docker.tar.gz
```

To build, (re)create and start the container execute the following command.

```
$ docker-compose up -d
```

The default port numbers that the Dockerized NXLog Manager uses can be configured in the [docker-compose.yml](#). As an example, to access the web interface of NXLog Manager at port 9080 instead of the default 9090 modify [docker-compose.yml](#) as follows.

NOTE

docker-compose.yml

```
1    ports:
2      - "4041:4041"
3      - "9080:9090"
4    restart: always
```

NOTE

The NXLog Manager Docker container includes MySQL therefore there is no need for installing and configuring MySQL separately. Now, you may proceed with the [NXLog Manager configuration](#).

91.4. Configuring NXLog Manager for standalone mode

To operate in standalone mode, NXLog Manager requires MySQL or MariaDB v5.5.

91.4.1. Installing MySQL Server Debian or Ubuntu

Install the mysql-server package:

```
# apt-get install mysql-server
```

91.4.2. Installing MySQL Server Centos or RedHat 6

Install the mysql-server package:

```
# yum install mysql-server
```

Now you may proceed with the [Database Initialization](#) step.

91.4.3. Installing MariaDB Server Centos or RedHat 7

MariaDB has replaced MySQL as the default package on Centos and RedHat 7. MariaDB is a fork of MySQL and should work seamlessly in place of MySQL. Install the mariadb-server package:

```
# yum install mariadb-server
```

Now you may proceed with the [Database Initialization](#) step.

91.5. Configuring NXLog Manager for cluster mode

It is possible to run multiple instances of NXLog Manager so that a group of agents connect to a specific Manager instance and all agents can be managed at the same time from either NXLog Manager instance no matter which one they are connected to. This mode is referred to as distributed mode or cluster mode.

The following needs to be set in the [/opt/nxlog-manager/conf/nxlog-manager.conf](#) configuration file on each instance:

nxlog-manager.conf

```
1 INSTANCE_MODE=distributed-manager
```

The NXLog Manager instances communicate over JMS. For this please set the public IP address of the interface in

/opt/nxlog-manager/conf/jetty-env.xml and make sure to replace the value 127.0.0.1 set for JMSBrokerAddress with the public IP:

jetty-env.xml

```
<Set name="jmsBrokerAddress">10.0.0.42</Set>
```

To operate in clustered mode, NXLog Manager requires MariaDB Galera Cluster v5.5.

91.5.1. Installing MariaDB Galera Cluster on Debian or Ubuntu

There is a very good installation guide [here](#). The MariaDB Galera Cluster installation and configuration steps are summarized below.

Add the package repository:

```
# apt-get install python-software-properties  
# apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
```

For Debian Wheezy:

```
# add-apt-repository 'deb http://mirror3.layerjet.com/mariadb/repo/5.5/debian wheezy main'
```

For Ubuntu Precise:

```
# add-apt-repository 'deb http://mirror3.layerjet.com/mariadb/repo/5.5/ubuntu precise main'
```

Resynchronize the package index files:

```
# apt-get update
```

Install the packages:

```
# DEBIAN_FRONTEND=noninteractive apt-get install -y rsync galera mariadb-galera-server
```

Add the following to */etc/mysql/conf.d/galera.cnf*:

galera.cnf

```
1 [mysqld]  
2 #mysql settings  
3 binlog_format=ROW  
4 default-storage-engine=innodb  
5 innodb_autoinc_lock_mode=2  
6 query_cache_size=0  
7 query_cache_type=0  
8 bind-address=0.0.0.0  
9 #galera settings  
10 wsrep_provider=/usr/lib/galera/libgalera_smm.so  
11 wsrep_cluster_name="my_wsrep_cluster"  
12 wsrep_cluster_address="gcomm://<IP1>, <IP2>, ..., <IPN>"  
13 wsrep_sst_method=rsync
```

Here IP1,...,IPN are the addresses of all nodes in the galera cluster. Distribute this file to all nodes.

Start the galera cluster:

First stop all nodes:

On node1:

```
# service mysql stop
```

On node2:

```
# service mysql stop
```

On nodeN:

```
# service mysql stop
```

Start the central node:

On node1:

```
# service mysql start --wsrep-new-cluster
```

Then start on all other nodes:

On node2:

```
# service mysql start
[ ok ] Starting MariaDB database server: mysqld . . . .
[info] Checking for corrupt, not cleanly closed and upgrade needing tables..
```

On nodeN:

```
# service mysql start
[ ok ] Starting MariaDB database server: mysqld . . . .
[info] Checking for corrupt, not cleanly closed and upgrade needing tables..
```

Verify all nodes are running:

```
# mysql -u root -e 'SELECT VARIABLE_VALUE as "cluster size" FROM INFORMATION_SCHEMA.GLOBAL_STATUS
WHERE VARIABLE_NAME="wsrep_cluster_size"'
```

This command should return N, i.e. the number of cluster nodes.

91.5.2. Installing MariaDB Galera Cluster on RedHat

There is an installation guide [here](#). The MariaDB Galera Cluster installation and configuration steps are summarized below.

To add the MariaDB repository create the file `/etc/yum.repos.d/mariadb.repo` with the following content:

`mariadb.repo`

```
1 [mariadb]
2 name = MariaDB
3 baseurl = http://yum.mariadb.org/5.5/centos6-amd64
4 gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
5 gpgcheck=1
```

Now install MariaDB and Galera:

```
# yum install MariaDB-Galera-server MariaDB-client galera
```

You can download and install 'socat' from <https://fedoraproject.org/wiki/EPEL> in case you are getting the following error:

```
Error: Package: MariaDB-Galera-server-5.5.40-1.el6.x86_64 (mariadb)
      Requires: socat
You could try using --skip-broken to work around the problem
```

To create an initial MariaDB configuration execute these commands and follow the instructions:

```
# service mysql start
# mysql_secure_installation
# mysql -u root -p
MariaDB [(none)]> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'password' WITH GRANT OPTION;
MariaDB [(none)]> FLUSH PRIVILEGES;
MariaDB [(none)]> exit
# service mysql stop
```

On each cluster node edit `/etc/my.cnf.d/server.cnf` and make sure to add the following content:

`server.cnf`

```
1 [mysqld]
2 pid-file      = /var/lib/mysql/mysqld.pid
3 port          = 3306
4
5 [mariadb]
6 query_cache_size=0
7 binlog_format=ROW
8 default_storage_engine=innodb
9 innodb_autoinc_lock_mode=2
10 wsrep_provider=/usr/lib64/galera/libgalera_smm.so
11 wsrep_cluster_address=gcomm://IP2,...,IPN
12 wsrep_cluster_name='cluster1'
13 wsrep_node_address='IP1'
14 wsrep_node_name='db1'
15 wsrep_sst_method=rsync
16 wsrep_sst_auth=root:password
```

Make sure to set the values appropriately on each node.

Start the cluster node using following command:

```
# /etc/init.d/mysql bootstrap
Bootstrapping the cluster.. Starting MySQL.... SUCCESS!
```

On the other nodes start it with the following command:

```
# service mysql start
Starting MySQL.... SUCCESS!
```

SELinux may block MariaDB from binding on the cluster port and it will print the following error in the MariaDB error log:

MariaDB Error

```
140805  7:56:00 [Note] WSREP: gcomm: bootstrapping new group 'cluster1'↵
140805  7:56:00 [ERROR] WSREP: Permission denied↵
140805  7:56:00 [ERROR] WSREP: failed to open gcomm backend connection: 13: error
while trying to listen 'tcp://0.0.0.0:4567?socket.non_blocking=1'↵
', asio error 'Permission denied': 13 (Permission denied)↵
```

NOTE

This can be solved by running

```
# setenforce 0
```

and setting

config

```
1 SELINUX=permissive
```

in '/etc/selinux/config'.

Now repeat the above installation steps on each node.

91.6. Database initialization

The NXLog Manager needs its initial configuration data to be loaded into the configuration database.

NOTE

If you are installing NXLog Manager in clustered mode, this only needs to be executed once for the DB cluster - i.e. only on the first node.

If you have a root password set for the MySQL/MariaDB database, please edit `/opt/nxlog-manager/db_init/my.cnf` and add the password to the password variable:

my.cnf

```
1 [client]
2 password=
```

Execute db initialization script (only once for the Galera cluster!):

```
$ cd /opt/nxlog-manager/db_init
# ./dbinit.sh
```

NOTE

To ensure that the MySQL/MariaDB database is started on boot on Centos/RedHat distributions, execute the following command:

```
# chkconfig mysql on (or chkconfig mariadb on)
```

Now start the nxlog-manager service and check the logs under `/opt/nxlog-manager/logs` if you are having trouble accessing the web interface at <http://x.x.x.x:9090/nxlog-manager>

The size of the maximum packet allowed by MySQL/MariaDB can be raised by adding the following to the global configuration options, typically `/etc/my.cnf` or `/etc/mysql/my.cnf`.

my.cnf

TIP

```
1 [mysqld]
2 max_allowed_packet = 256M
```

Raising the size of the maximum allowed packet will eliminate any *max_allowed_packet exceeded* error messages from the log files.

91.7. NXLog Agent installation

91.7.1. Installing on Debian Wheezy

```
# dpkg -i nxlog_X.X.XXXX_amd64.deb
# apt-get -f install
```

91.7.2. Installing on Redhat

To install the NXLog agent on RedHat you need to issue the following command:

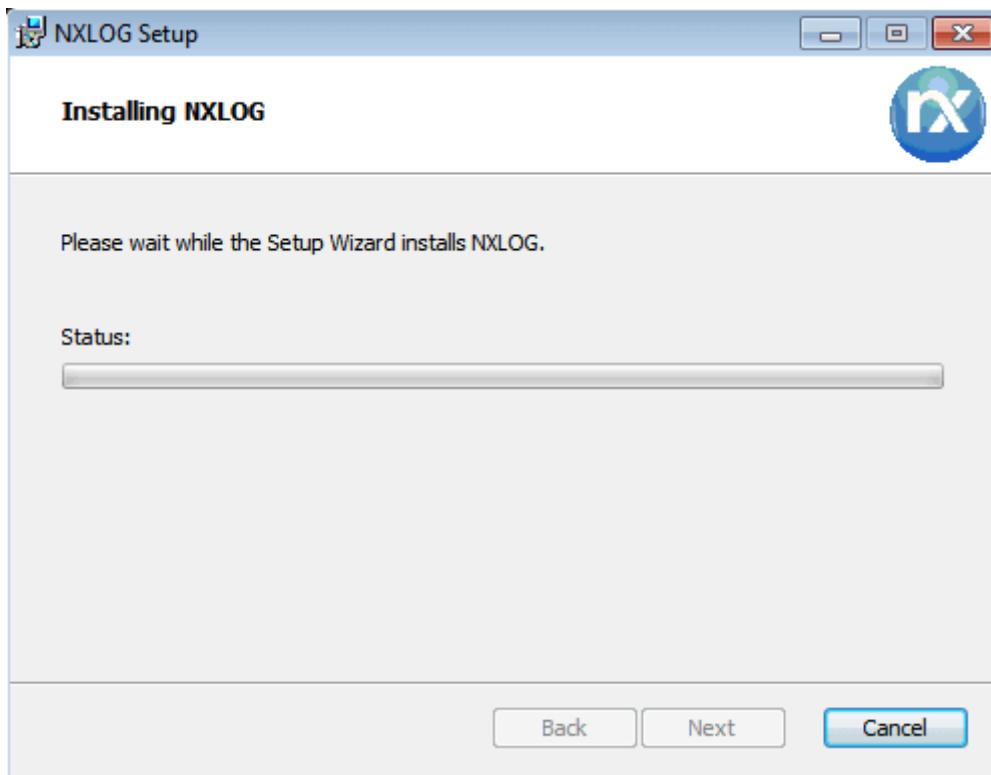
```
# rpm -ivh nxlog-X.X.XXXX-1.x86_64.rpm
```

Depending on the package there may be additional dependencies required to be installed:

```
# yum install dialog apr perl perl-DBI perl-JSON openssl pcre zlib expat libcap libdbi
```

91.7.3. Installing on Windows

On windows simply run the msi installer. You should be greeted by the following screens:



Simply click next, accept the license agreement then finish the installation.

It is possible to automate the installation on Windows using `msiexec`:

```
msiexec /i nxlog-xxx.msi /quiet
```

The msi can be also installed from group policy.

91.8. NXLog Manager configuration

Once the nxlog-manager service is running you should be able to access the web interface at <http://x.x.x.x:9090/nxlog-manager> The default access credentials are as follows:

```
User ID: admin  
Password: nxlog123
```

91.8.1. Installation Wizard

When the administrator logs in the first time a dialog window will be displayed to help with the initial configuration: 'You don't have a default CA and AgentManager certificate. Do you want to create a CA and a CERT?' Click Yes to proceed with the CA setup.

Create default CA (1/3)

On this page you can create a new default CA.

*Name:	CA
Country	country
State	state
City	city
Organization	organization
Org. Unit	organization unit
Valid To	2019-09-24
Set as default CA	<input checked="" type="checkbox"/>
Use parameters as default	<input checked="" type="checkbox"/>

Create **Help**

Fill in the form and then click Create.

The next dialog window will ask to create a certificate for the Agent manager.

Create Agent manager certificate (2/3)

On this page you can create a certificate for the Agent manager.

*Name:	AGENT_MANAGER
Country	country
State	state
City	city
Organization	organization
Org. Unit	organization unit
Valid To	2019-09-24

Create **Help**

Fill in the form and then click Create.

Finally the Agent manager settings need to be provided.

Agent manager settings (3/3)

On this page you can set basic parameters for the Agent manager.

<input checked="" type="checkbox"/> Accept agent connections
*Listen address: 0.0.0.0
*Port: 4041
<input checked="" type="checkbox"/> Initiate connection to agents
<input checked="" type="checkbox"/> Don't encrypt agent manager's private key
<input checked="" type="checkbox"/> Start Agent manager

Finish **Help**

The default values should be sufficient for most users, click Finish.

The initial settings can be changed any time later under

'Admin → Settings → Agent Manager'

NOTE

and

'Admin → Settings → Certificates'.

91.8.2. Agent manager configuration

Navigate to 'Admin → Settings → Agent Manager' and fill out the following form accordingly.

NOTE

If you have already configured the Agent manager with the Wizard as described in the previous section then you will not need to modify anything here. Just make sure your settings are correct.

Select whether you would like the agents or the agent manager to initiate the connection. This can be useful when special firewall and zone rules apply. Make sure that the agent manager certificate is properly set. Click Save & Restart to apply settings.

Accept agent connections

***Listen address:** 0.0.0.0

***Port:** 4041

CA: n xm-ca

Certificate: AgentManager

Initiate connection to agents

Don't encrypt agent manager's private key

Subject Name Authorization

- Warn if untrusted.
- Reject agent.
- Disable.

Agent name

- Use reverse DNS name, else IP address.
- Use reverse DNS name, else certificate subject name.
- Use certificate subject name.

Status: Agent Manager listener is accepting connections on 0.0.0.0:4041
0 agents connected

Agent Manager connector not started.

Save & Restart **Help**

91.8.3. Connecting agents

In order to ensure that the NXLog agents can only be controlled by the NXLog Manager, the NXLog agents are controlled over a secure trusted SSL and each NXLog agent needs its own private key and certificate.

91.8.3.1. Automated deployment

The requirement of a private key and certificate pair for each NXLog agent would prevent automated installation. Fortunately it is possible to install the NXLog agents with only the CA certificate and an initial configuration containing only the details on how to establish the control connection with the NXLog Manager.

The installation steps for automated agent deployment consist of the following:

- Install the NXLog package (msi/rpm/deb).
- Copy the initial configuration file.
- Copy the CA certificate file.
- Start the NXLog service and verify that it is connected.

These steps are discussed below.

To export the CA certificate, please navigate to 'Admin → Certificates' and select the CA with the checkbox as shown in the screenshot below.

Certificates list

	Name	Type	Activation	Expiration	Status	Private Key	Search:
<input checked="" type="checkbox"/>	CA	CA	2014-10-24 16:21	2019-09-24 00:00	VALID	YES	
<input type="checkbox"/>	AGENT_MANAGER	CERT	2014-10-24 16:22	2019-09-24 00:00	VALID	YES	

Showing 1 to 2 of 2 entries

[Add new CA](#) [Create certificate](#) [Import](#) [Export](#) [Revoke](#) [Renew](#) [Delete](#) [Help](#)

Click Export. The CA certificate should be exported using the 'Certificate in PEM format' option. Save the file as **CA-cert.pem**.

The default installation of NXLog will create a file that is to be updated by NXLog Manager. On Windows systems this is **C:\Program Files (x86)\nxlog\conf\log4ensics.conf**, on GNU/Linux systems this configuration file is **/opt/nxsec/var/lib/nxlog/log4ensics.conf**. When doing an automated deployment, this file should be replaced with the following default configuration.

log4ensics.conf

```

LogLevel INFO<
LogFile %LOGDIR%/nxlog.log<
<

<Extension agent_management><
    Module      xm_soapadmin<
    Connect     X.X.X.X<
    Port        4041<
    SocketType SSL<
    CAFile      %CERTDIR%/CA-cert.pem<
    AllowUntrusted FALSE<
    RequireCert TRUE<
    <ACL conf><
        Directory  %CONFDIR%<
        AllowRead  TRUE<
        AllowWrite TRUE<
    </ACL><
    <ACL cert><
        Directory  %CERTDIR%<
        AllowRead  TRUE<
        AllowWrite TRUE<
    </ACL><
</Extension><

```

Please make sure to replace X.X.X.X with the proper IP address of the NXLog Manager instance that the NXLog agent needs to be connected to.

The CA certificate file **CA-cert.pem** must be also copied to the proper location as referenced in the above configuration which is normally under **C:\Program Files (x86)\nxlog\cert** on Windows systems and under **/opt/nxsec/var/lib/nxlog/cert/** on GNU/Linux systems.

NOTE

When the configuration and certificate files are updated remotely, NXLog must have permissions to overwrite these files when it is running as a regular (i.e. nxlog) user. Please make sure that the ownership is correct:

```
# chown -R nxlog:nxlog /opt/nxsec/var/lib(nxlog)
```

Now start the NXLog service. The **nxlog.log** file should contain the following if the NXLog agent has successfully connected.

nxlog.log

```
2014-10-24 17:24:46 WARNING no functional input modules!←
2014-10-24 17:24:46 WARNING no routes defined!←
2014-10-24 17:24:46 INFO nxlog-2.8.1281 started←
2014-10-24 17:24:46 INFO connecting to agent manager at X.X.X.X:4041←
2014-10-24 17:24:46 INFO successfully connected to agent manager at X.X.X.X:4041 in SSL mode←
```

Click the 'AGENTS' menu to see the list of agents. You should see the newly connected agent with an UNTRUSTED (yellow) status. If you don't see the agent there, check the logs for error diagnostics.

The screenshot shows the 'Agent list' tab of the NXLog Manager. The table displays one agent entry:

Status	Agent name	Version	Host	Started	Load	Mem. usage	Received	Received today	Processing	Sent	Sent today
●	192.168.56.102	2.8.1249	192.168.56.102	2015-05-19 17:10	0.04	4.42M	0	0	0	0	0

Below the table are several buttons: Refresh status, Start, Stop, Update config, Reload, Configure, Add, Delete, Clone, Download config, View log, Issue certificate, Renew certificate, and Help.

The name of the untrusted agent should be the reverse DNS of its IP address.

In order to establish a mutually trusted connection between the NXLog agent and the Agent manager a certificate and private key pair needs to be issued and transferred to the agent. Select the untrusted agent in the list and click Issue certificate. When 'Update connected agents' is enabled, the newly issued certificate and the configuration will be pushed to the agent. The agent will need to reload the configuration in order to reconnect with the certificate, select the agent and click Reload.

After the agent has sucessfully reconnected and the agent list is refreshed the agent status should be 'online' showing a green ball.

The screenshot shows the 'Agent list' tab of the NXLog Manager. The table displays the same agent entry, but the status is now green (online). The rest of the interface is identical to the previous screenshot.

At this stage the NXLog agent should be operational and can now be [managed and configured](#) from the NXLog Manager interface.

91.8.3.2. Manual deployment

Manual deployment requires adding an agent using Add on the interface. After the agent is configured and has its certificate issued, select its checkbox in the agent list and click Download config.

The screenshot shows the 'Agent list' tab in the NXLog Manager. It displays a table of agents with columns for Status, Agent name, Version, Host, Started, Load, Mem. usage, Received, Received today, Processing, Sent, and Sent today. The 'Status' column uses color-coded icons to indicate the agent's status: green for online, grey for offline. The 'Received' and 'Sent' columns include progress bars. At the bottom of the table, there are buttons for Refresh status, Start, Stop, Update and reload, Configure, Add, Delete, Clone, Download config, View log, Assign template, Issue certificate, Renew certificate, and Help.

On GNU/Linux systems extract the agents-config.zip and put the files under /opt/nxsec/var/lib(nxlog. Make sure the files have the proper ownership:

```
# chown -R nxlog:nxlog /opt/nxsec/var/lib/nxlog
```

On Windows systems place the certificates in C:\Program Files (x86)\nxlog\cert. After restarting the NXLog service you should now see your agent as 'online' under AGENTS.

91.8.4. Configuring agents

Once the agent is connected and is shown as 'Online' it can be remotely configured from the NXLog Manager web interface. To configure the log collection, click on your agent in the agent list and then select the 'Configure' tab. Click 'Routes' and add a route. Add a tcp input module for testing purposes:

```
Name: tcptest
Module: TCP Input (im_tcp)
Listen On: 0.0.0.0
Port: 1514
Input Format: line based
```

The screenshot shows the 'Add module' dialog for a TCP Input module. It has tabs for 'Parameters' (selected) and 'Expert'. Under 'Common parameters', fields are filled with: *Name: 'tcpin', *Module: 'TCP Input (im_tcp)'. Under 'Module specific parameters', fields are filled with: *Listen on: '0.0.0.0', *Port: '1514', Input Format: 'line based'. At the bottom are 'Cancel', 'Save' (highlighted in blue), and 'Help' buttons.

Add an output module. For test purposes we will use a null output that discards the data.

```
Name: out
Module: Null Output (om_null)
```

Now click 'Update config' on the 'Info' tab, then click 'Reload'. After the agent is restarted you should now see the newly configured modules on the 'Modules' tab.

Test the data collection:

```
telnet x.x.x.x 1514
type something
```

On the Modules tab check all modules and click 'Refresh status'. The count under the Received column should be (at least) 1.

The system should now be ready to be further configured as per your requirements.

91.8.5. Configuring logger settings

The NXLog Manager keeps by default log files in the `/opt/nxlog-manager/log` directory. Log priorities, levels as well as log rotation can be configured as per your requirements in the `/opt/nxlog-manager/conf/log4j.xml` file. The default configuration will create two separate files `nxlog-manager.log` and `nxlog-manager.err` where only information level messages will be logged on the first file and error level messages will be logged on the second file. Log rotation is set by default to rotate both files at the beginning of each month. The frequency of log rotation can be controlled by the `DatePattern` parameter, as shown below.

`log4j.xml`

```
<appender name="internalAppender" class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File" value="${logs.root}.log"/>
    <param name="Threshold" value="INFO"/>
    <param name="DatePattern" value=".yyyy-MM"/>
    <layout class="com.nxsec.log4ensics.common.logging.ContextPatternLayout">
        <param name="ConversionPattern" value="%d %p $host $user $component [%c] - %m %n"/>
    </layout>
</appender>
```

The following table summarizes different `DatePattern` options.

Table 59. Table DatePattern options

DatePattern	Rollover schedule
'.yyyy-MM	Rollover at the beginning of each month
'.yyyy-ww	Rollover at the first day of each week. The first day of the week depends on the locale.
'.yyyy-MM-dd	Rollover at midnight each day.
'.yyyy-MM-dd-a	Rollover at midnight and midday of each day.
'.yyyy-MM-dd-HH	Rollover at the top of every hour.
'.yyyy-MM-dd-HH-mm	Rollover at the beginning of every minute.

NOTE

The `/opt/nxlog-manager/conf/log4j.xml` defines three different files, the two mentioned above as well as a debug file that needs to be enabled separately. Log rotation can be controlled individually for each log file by altering the `DatePattern` parameter at each of the three appender sections.

To enable the debug logging:

- change the priority level from `INFO` to `DEBUG`,
- change `WARN` level to `DEBUG` in the loggers you require,
- remove the comment from the `debugAppender` reference, as shown bellow.

log4j.xml

```
<root>
    <priority value="DEBUG"/>
    <appender-ref ref="internalAppender"/>
    <appender-ref ref="errorAppender"/>
    <appender-ref ref="debugAppender"/>
</root>
```

91.9. Enabling HTTPS for NXLog Manager

Depending on the version of NXLog Manager installed, follow the instructions below.

91.9.1. NXLog Manager version 4.X.XXX

To enable HTTPS for secure connections you need to uncomment the last section in `<NXLogManager_HOME>/conf/jetty-config.xml` which looks as follows:

jetty-config.xml

```
<!--
    <New id="sslContextFactory"
        class="com.nxsec.log4ensics.dbmanager.common.server.util.ssl.SslContextFactory">
        <Set name="ServerCertificate"><Property name="jetty.home" default=".." />/conf/jetty8-
        cert.pem</Set>
        <Set name="ServerKey"><Property name="jetty.home" default=".." />/conf/jetty8-key.pem</Set>
        <Set name="ServerKeyPassword"></Set>
    </New>

    <Call name="addConnector">
        <Arg>
            <New class="${sslConnector}">
                <Arg><Ref id="sslContextFactory" /></Arg>
                <Set name="Port">9443</Set>
                <Set name="maxIdleTime">30000</Set>
                <Set name="Acceptors">2</Set>
                <Set name="AcceptQueueSize">100</Set>
            </New>
        </Arg>
    </Call>
-->
```

To disable plain HTTP you should comment out the following connector section in `jetty-config.xml`:

jetty-config.xml

```
<!--
<Call name="addConnector">
    <Arg>
        <New class="org.eclipse.jetty.server.nio.SelectChannelConnector">
            <Set name="host"><Property name="jetty.host" /></Set>
            <Set name="port"><Property name="jetty.port" default="9090"/></Set>
            <Set name="maxIdleTime">300000</Set>
            <Set name="Acceptors">2</Set>
            <Set name="statsOn">false</Set>
            <Set name="confidentialPort">9443</Set>
            <Set name="lowResourcesConnections">20000</Set>
            <Set name="lowResourcesMaxIdleTime">5000</Set>
        </New>
    </Arg>
</Call>
-->
```

91.9.2. NXLog Manager version 5.X.XXX

To enable HTTPS for secure connections you need to uncomment three sections in `<NXLogManager_HOME>/conf/jetty-config.xml` which look as follows:

jetty-config.xml

```
<New id="sslContextFactory"
class="com.nxsec.log4ensics.dbmanager.common.server.util.ssl.SslContextFactory">
    <Set name="ServerCertificate"><Property name="jetty.home" default=".." />/conf/jetty9-
cert.pem</Set>
    <Set name="ServerKey"><Property name="jetty.home" default=".." />/conf/jetty9-key.pem</Set>
    <Set name="ServerKeyPassword"></Set>
    <Set name="EndpointIdentificationAlgorithm"></Set>
    <Set name="NeedClientAuth"><Property name="jetty.ssl.needClientAuth" default="false"/></Set>
    <Set name="WantClientAuth"><Property name="jetty.ssl.wantClientAuth" default="false"/></Set>
    <Set name="ExcludeCipherSuites">
        <Array type="String">
            <Item>SSL_RSA_WITH_DES_CBC_SHA</Item>
            <Item>SSL_DHE_RSA_WITH_DES_CBC_SHA</Item>
            <Item>SSL_DHE_DSS_WITH_DES_CBC_SHA</Item>
            <Item>SSL_RSA_EXPORT_WITH_RC4_40_MD5</Item>
            <Item>SSL_RSA_EXPORT_WITH_DES40_CBC_SHA</Item>
            <Item>SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA</Item>
            <Item>SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA</Item>
            <Item>SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA</Item>
            <Item>SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA</Item>
            <Item>TLS_DHE_RSA_WITH_AES_256_CBC_SHA256</Item>
            <Item>TLS_DHE_DSS_WITH_AES_256_CBC_SHA256</Item>
            <Item>TLS_DHE_RSA_WITH_AES_256_CBC_SHA</Item>
            <Item>TLS_DHE_DSS_WITH_AES_256_CBC_SHA</Item>
            <Item>TLS_DHE_RSA_WITH_AES_128_CBC_SHA256</Item>
            <Item>TLS_DHE_DSS_WITH_AES_128_CBC_SHA256</Item>
            <Item>TLS_DHE_RSA_WITH_AES_128_CBC_SHA</Item>
            <Item>TLS_DHE_DSS_WITH_AES_128_CBC_SHA</Item>
        </Array>
    </Set>
</New>
```

jetty-config.xml

```
<New id="sslHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
    <Arg><Ref refid="httpConfig"/></Arg>
    <Call name="addCustomizer">
        <Arg><New class="org.eclipse.jetty.server.SecureRequestCustomizer"/></Arg>
    </Call>
</New>
```

jetty-config.xml

```
<Call name="addConnector">
    <Arg>
        <New id="sslConnector" class="org.eclipse.jetty.server.ServerConnector">
            <Arg name="server"><Ref refid="Server" /></Arg>
            <Arg name="factories">
                <Array type="org.eclipse.jetty.server.ConnectionFactory">

                    <!-- uncomment to support proxy protocol
                    <Item>
                        <New class="org.eclipse.jetty.server.ProxyConnectionFactory" />
                    </Item>-->

                    <Item>
                        <New class="org.eclipse.jetty.server.SslConnectionFactory">
                            <Arg name="next">http/1.1</Arg>
                            <Arg name="sslContextFactory"><Ref refid="sslContextFactory"/></Arg>
                        </New>
                    </Item>
                    <Item>
                        <New class="org.eclipse.jetty.server.HttpConnectionFactory">
                            <Arg name="config"><Ref refid="sslHttpConfig" /></Arg>
                        </New>
                    </Item>
                </Array>
            </Arg>

            <Set name="host"><Property name="jetty.host" /></Set>
            <Set name="port"><Property name="jetty.https.port" default="9443" /></Set>
            <Set name="idleTimeout"><Property name="ssl.timeout" default="30000" /></Set>
        </New>
    </Arg>
</Call>

<Call class="java.lang.System" name="setProperty">
    <Arg>org.apache.jasper.compiler.disablejsr199</Arg>
    <Arg>true</Arg>
</Call>

<!-- Fix for java.lang.IllegalStateException: Form too large 207624>200000 -->
<Call name="setAttribute">
    <Arg>org.eclipse.jetty.server.Request.maxFormContentSize</Arg>
    <Arg><Property name="jetty.maxFormContentSize" default="1000000" /></Arg>
</Call>
```

91.9.3. NXLog Manager SSL keys

All versions of NXLog Manager are bundled with default key pair in pem format to be used for the secure connection under `<NXLogManager_HOME>/conf/`, namely `jetty8-cert.pem` and `jetty8-key.pem`. These can be customized in `jetty-config.xml` by editing `ServerCertificate` and `ServerKey` properties of

`sslContextFactory`. Provide the `ServerKeyPassword` if the private key is password protected.

Now NXLog Manager can be restarted with HTTPS enabled with the default port 9443. The port can also be also customized in `jetty-config.xml`.

91.10. Upgrading NXLog Manager

Upgrading from earlier versions of NXLog Manager will require changes to the database structure. To complete the upgrade it is required to stop the NXLog Manager service before proceeding.

```
# service nxlog-manager stop
```

NOTE

It is always advisable and good practice to create a backup before upgrading. This enables the process to be rolled back if something goes wrong. Use `mysqldump` or `phpMyAdmin` to backup MySQL/MariaDB.

91.10.1. Upgrade to version 4.0.6223 or later

Follow this procedure, if you are running a version of NXLog Manager, earlier than 4.0.6223 and you are planning to upgrade to version 4.0.6223 or later, but not version 5.

After stopping the NXLog Manager service, upgrade NXLog Manager but do not start the service. Navigate to `/opt/nxlog-manager/db_init/upgrade/` and execute the command:

```
# mysql -u root nxlog-manager4 < upgrade_to_6223.sql
```

The upgraded version of NXLog Manager service can now be started.

91.10.2. Upgrade from version 4 to version 5

Follow this procedure, to upgrade from version 4 of NXLog Manager to version 5.

After stopping the NXLog Manager service, upgrade NXLog Manager but do not start the service. The upgraded NXLog Manager requires first a **database initialization**. Do not start the NXLog Manager service as part of the initialization. After initializing the database, navigate to `/opt/nxlog-manager/db_init/upgrade/` and execute the command:

```
# mysql -u root -p < upgrade_v4_to_v5.sql
```

The command will copy all the relevant information from the earlier version of NXLog Manager database to the new database without altering the old database. The upgraded version of NXLog Manager service can now be started.

91.11. Host Setup Common Issues

This section describes some common issues, that may prevent NXLog Manager from working correctly, but are not related to the installation or configuration of the Manager itself and how to remedy those issues.

91.11.1. Hostname resolve issues

For NXLog Manager to work correctly, the hostname of the host must resolve to the correct IP address. The following error will be present in `/opt/nxlog-manager/log/nxlog-manager.err`, if that is not the case.

nxlog-manager.err

```
2016-11-21 16:14:31,015 ERROR localhost unknown log4ensics [net.sf.ehcache.Cache] - Unable to set
localhost. This prevents creation of a GUID. Cause was:
java.net.UnknownHostException: nxlogmgr.domain.local
```

To set the hostname to *myname* you can either issue the command:

```
# hostname myname
```

or edit **/etc/hostname** directly. In both cases the hostname will change on the next reboot. For the hostname to resolve correctly, there should be a matching line in **/etc/hosts** that usually aliases the host name and the FQDN.

/etc/hosts

```
1 127.0.1.1 myname.example.com myname
```

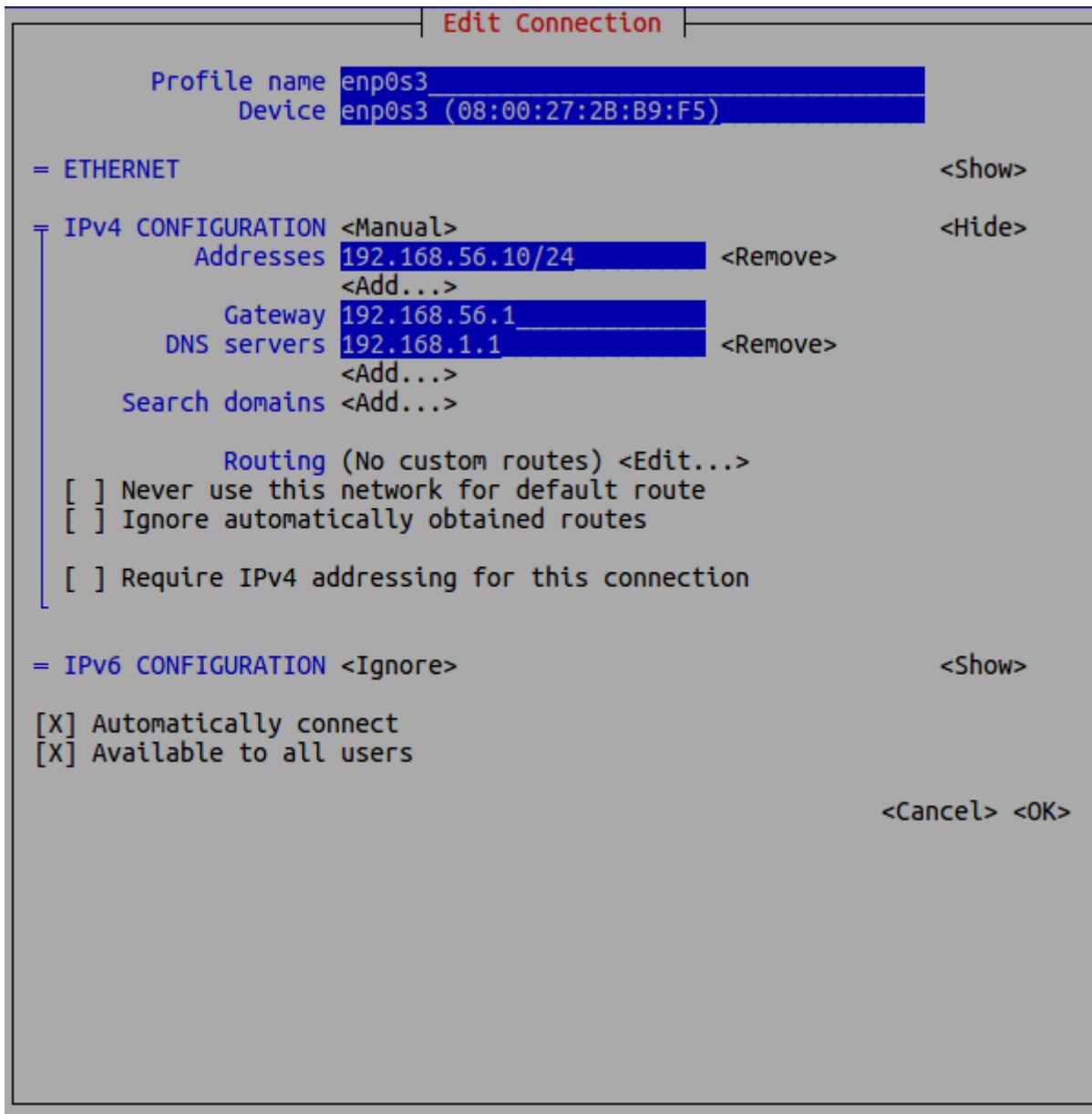
91.11.2. DNS lookup issues

Make sure that your DNS is setup correctly and it is functioning properly. DNS timeouts and errors can cause various issues, mainly because TLS certificate verification is using DNS lookups.

If your Linux Distribution or your setup uses Network Manager, DNS can be configured by issuing the following command.

```
# nmtui
```

Select the 'Edit a connection' option and then 'Edit connection'. You will presented with a dialog similar to the following, where DNS servers can be added, altered or removed.



If your setup does not use the Network Manager, you must edit the `/etc/resolv.conf` file manually. Usually up to three nameservers can be setup using the `nameserver` keyword, followed by the IP address of the nameserver.

To test whether your configuration functions correctly, use the `host` or `dig` programs to perform both a DNS lookup and a reverse DNS lookup (by querying with an IP address). Make sure that participating hosts on your NXLog collection system are resolved correctly.

Chapter 92. Dashboard and the Menu

92.1. Logging in

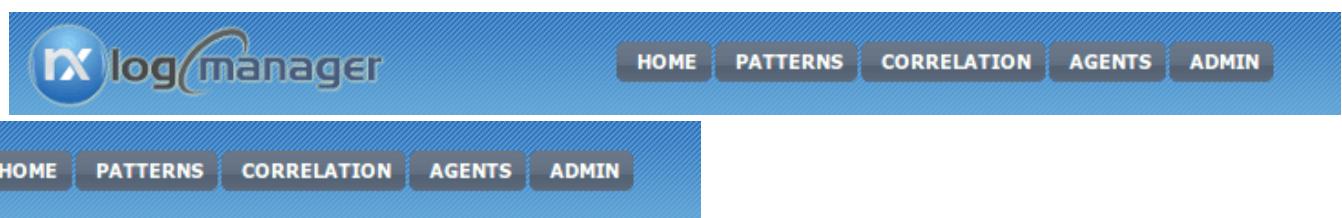
After the NXLog Manager web application is installed successfully, you should be able to see the following login screen after entering the URL of the application in the browser:

A screenshot of the NXLog Manager login page. It features a large silver padlock icon on the left. The word 'Login' is centered above two input fields. The first field is labeled 'User ID:' with 'admin' typed into it. The second field is labeled 'Password:' with a series of dots representing the password. Below the fields is a blue 'Login' button.

The default user id and password is admin/nxlog123. This should be changed as soon as possible.

92.2. The menu bar

The menu bar is located in the header of all pages and looks like the following:



HOME

Clicking on this menu item will load the [dashboard](#).

PATTERNS

CREATE PATTERN

Click this menu item to create a new pattern. See the [Patterns](#) chapter for more information about patterns.

LIST PATTERNS

Click this menu item to list all available patterns. See the [Patterns](#) chapter for more information about patterns.

SEARCH PATTERN

Click this menu item to search for specific patterns. See the [Patterns](#) chapter for more information about patterns.

CREATE GROUP

Click this menu item to create a new pattern group. See the [Patterns](#) chapter for more information about pattern groups.

LIST GROUPS

Click this menu item to list all available pattern groups. See the [Patterns](#) chapter for more information about pattern groups.

IMPORT PATTERN

Click this menu item to import a pattern database file. See the [Patterns](#) chapter for more information about patterns.

CREATE FIELD

Click this menu item to create a new field. See the [Fields](#) chapter for more information about fields.

LIST FIELDS

Click this menu item to list all available fields. See the [Fields](#) chapter for more information about fields.

CORRELATION

Click this menu item to manage correlation rules and rulesets. See the [Correlation](#) chapter for more information.

AGENTS

Click this menu item to see the list of nxlog agents and manage them. See the [Agents](#) chapter for more information.

ADMIN***USERS***

Clicking this menu item will load the user management interface. See the [Users](#) section for more information.

ROLES

Clicking this menu item will load the role management interface. See the [Roles](#) section for more information.

CERTIFICATES

Click this menu item to see the list of certificates available in the built in PKI. See the [Certificates](#) chapter for more information.

SETTINGS

Clicking this menu item allows to change various system-wide settings and personal preferences. See the [Settings](#) chapter for more information.

LOGOUT

Click this menu item to log out of the Log4ensics web application and terminate your session. You will be logged out automatically after some idle time if there is no activity.

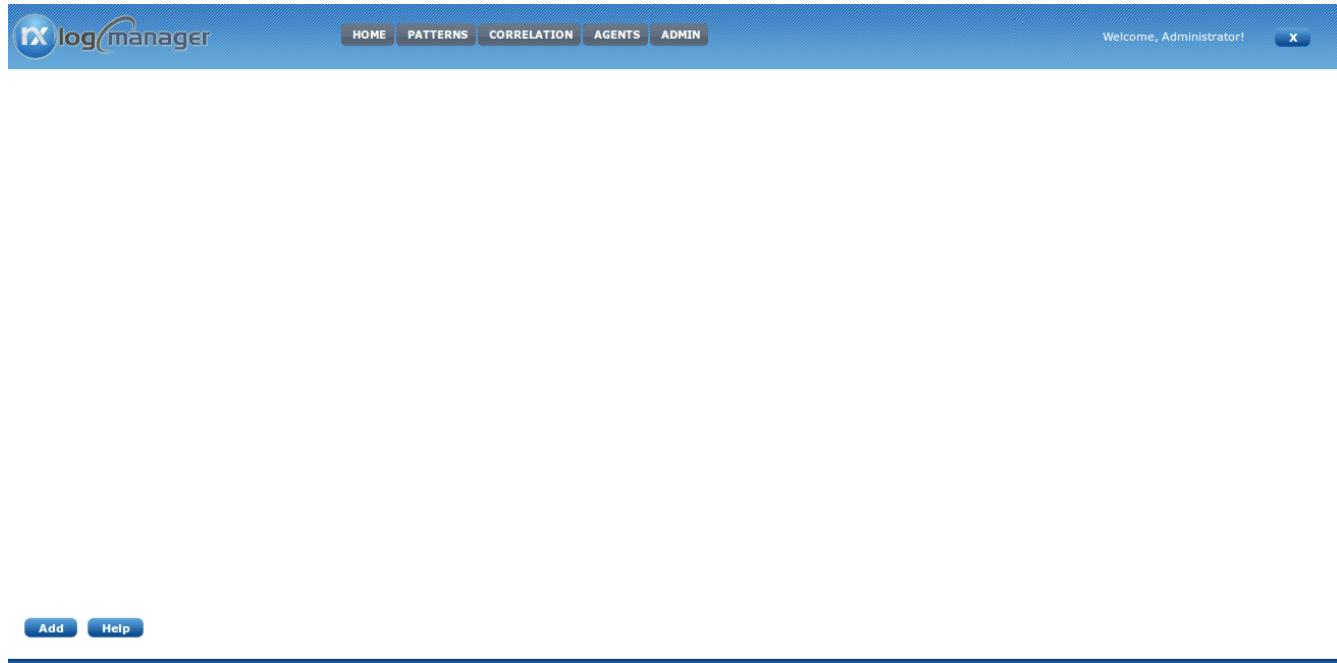
NOTE

Depending on your configured roles, some menu items will not be shown which you do not have access rights to. See the [Roles](#) section for more information about the access control in NXLog Manager.

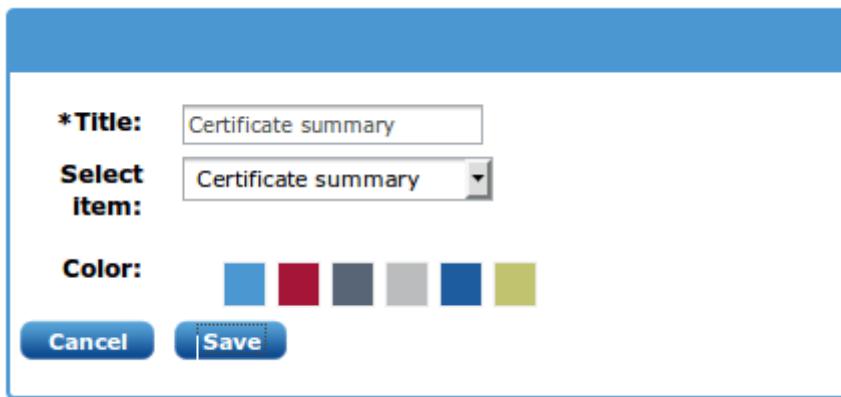
The successive chapters are organized to cover each of these components which can be accessed from the menu.

92.3. Dashboard

On the first login the following screen should appear.



The dashboard can be customized and will display different content for each logged in user. After clicking the [**Add**] button, an empty dashboard item will appear:



The following item types can be selected with the combo box:

Agent list

The number of agents are displayed for each category:

- Online
- Offline
- Error

- Unmanaged

See the <>nxlog_manager_agents,Agents>> chapter for more information about agent statuses.

Jobs summary

Will display a summary about scheduled jobs.

Certificate summary

Will display a summary about certificates grouping them by the following categories:

- Expired
- To be expired in the next 10 days
- Revoked
- Valid

See the <>nxlog_manager_certificates,Certificates>> chapter for more information.

Agent chart

Will display one of these agent charts for an agent:

- Load average and memory usage
- Overall event count
- Event count of a separate modules
- Module variables and statistical counters, when such are configured in agent statistics page

After the required parameters are filled in, click [**Save**] to add the item to the dashboard. Click [**Cancel**] if you wish to discard the dashboard item.

The header bar of the dashboard item can be clicked to drag and move the dashboard item around.



The following items are on the header bar from left to right order:

Up arrow

Click to minimize the dashboard item.

Title

The title which was filled in when editing the dashboard item.

Edit

Click to edit the dashboard item.

X

Click the X button to remove the dashboard item.

Down arrow

Click the down arrow in the top right corner to maximize the dashboard item.

Chapter 93. Fields

There is one thing common in all event log messages. All contain important data such as user names, IP addresses, application names, etc. This way an event can be represented as a list of key-value pairs which we call a "field". The name of the field is the key and the field data is the value. In another terminology this meta-data is sometimes referred to as event property or message tag.

As NX-Log4ensics and NXLOG operate with a set of fields belonging to a log message, it is important to manage these as well. Fields in NX-Log4ensics are typed, this allows complex operations and efficient storage of event log data. All of the major components depend on fields and these are used in various places in Log4ensics, including [Patterns](#), [Correlation](#) and [Agent configuration](#).

Log4ensics comes with a set of predefined fields which should be enough for the general cases but can be extended to suit custom requirements. To list the available fields, click on the [LIST FIELDS](#) menu item under the PATTERN menu. A list similar to the following should appear:

Fields

	Field	Field Type	Field Description	Field Persist	Field Lookup
<input type="checkbox"/>	AccountAuditID	INTEGER	The unique identifier corresponding to the account performing the event	true	false
<input type="checkbox"/>	AccountEffectiveGroupID	INTEGER		true	false
<input type="checkbox"/>	AccountEffectiveGroupName	STRING	The name of the primary group associated with the effective user	true	true
<input type="checkbox"/>	AccountEffectiveID	INTEGER	The effective user ID (UID)	true	false
<input type="checkbox"/>	AccountEffectiveName	STRING	The effective user name	true	true
<input type="checkbox"/>	AccountGroupID	INTEGER	The ID of the group(s) to which the user belongs	true	false
<input type="checkbox"/>	AccountGroupName	STRING	The group(s) to which the user account belongs	true	true
<input type="checkbox"/>	AccountID	INTEGER	The unique identifier assigned to the user account, often called the user id (uid)	true	false
<input type="checkbox"/>	AccountName	STRING	The name associated with the user account	true	true
<input type="checkbox"/>	AccountRole	STRING	The role assigned to the user's account. Used for role-based access control (RBAC) and in systems such as Security Enhanced (SE) Linux	true	true
<input type="checkbox"/>	AccountType	STRING	The type of the account	true	true

Showing 1 to 11 of 201 entries

1
2
3
4
5

[Create](#)
[Edit](#)
[Delete](#)
[View](#)
[Help](#)

The field properties will be explained shortly as we look at creating and modifying fields. To do this, click on [[Create](#)] or [[Edit](#)] under the field list.

Field

Properties

Name

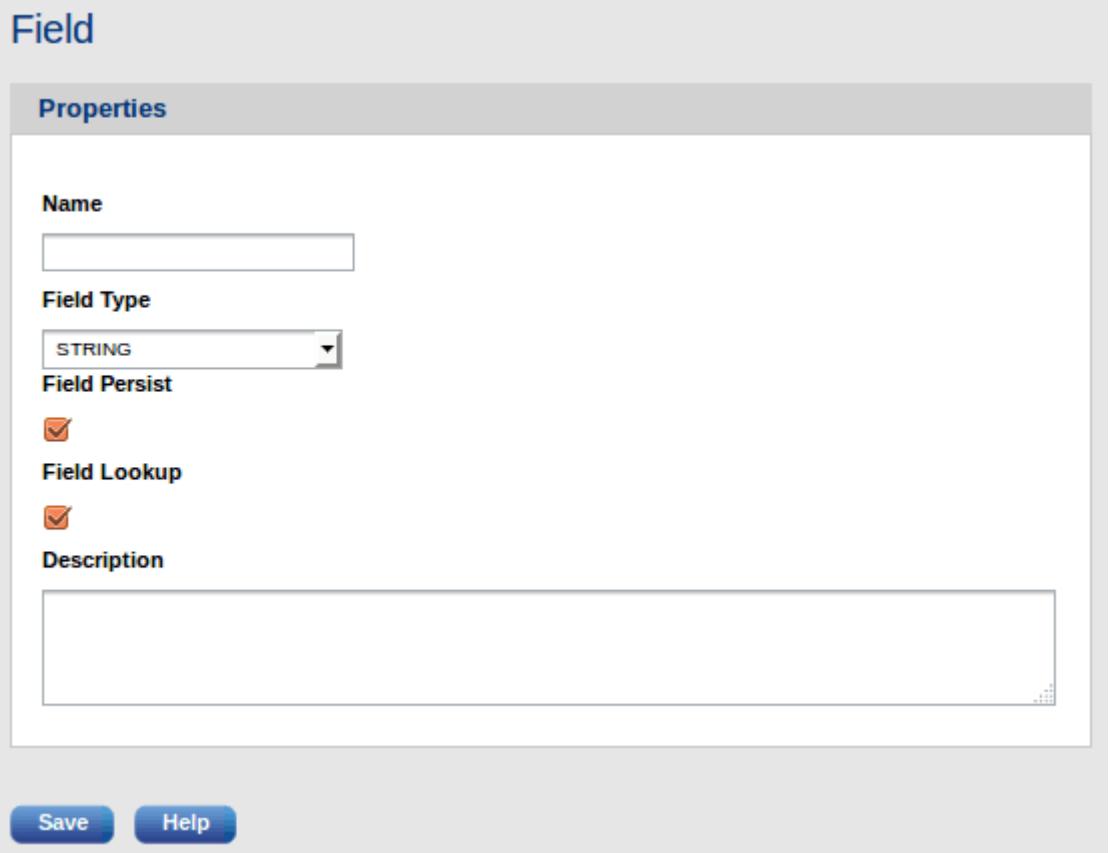
Field Type
▼

Field Persist

Field Lookup

Description

Save **Help**



The field properties are as follows:

Name

The name of the field will be used to refer to the field from various places in Log4ensics and nxlog.

Type

The following types can be chosen for a field:

- STRING
- INTEGER
- BINARY
- DATETIME
- IPV4ADDR
- IPV6ADDR
- BOOLEAN

Persist

If this option is not enabled, field values will be only available to the nxlog agent for correlation, pattern matching. Fields should be persisted if the information is needed in additional functions.

Lookup

This is a special property and only takes effect when the field is persistent and is a string type. The lookup property should be enabled for fields whose values are highly repetitive such as user names, enumerations, host names etc. This enables the storage engine to map the value to an integer which yields significant compression and performance boost.

Description

The description is only used as an information about the field.

The field list is kept in the configuration database.

Chapter 94. Patterns

Patterns provide a way to extract important information (e.g. user names, ip addresses, urls etc.) from free-form log messages. Many sources generate such free-from log messages where this information is contained within a human readable sentence or message, for example Syslog.

Consider the following example generated by the SSH server when an authentication failure occurs:

```
Failed password for john from 127.0.0.1 port 1542 ssh2
```

To be able to create a report about authentication failures, the username (john in the above example) needs to be extracted. Regular expressions are commonly used for this purpose.

The patterns used by NXLog Manager and NXLog are special in a way that these are not only single regular expressions.

- Patterns contain match criteria to be executed against one or more fields. A pattern matches only if all fields with a match criteria match. This technique allows patterns to be used with structured logs as well.
- The matching executed against the field(s) can be an exact match or a regular expression.
- Patterns can extract data from strings using captured substrings and store these in separate fields.
- Patterns can modify the log by setting additional fields. This is useful for message classification.
- Patterns can contain test cases for validation.
- Patterns are grouped under Pattern Groups.

Patterns are used by the NXLog agent. This makes it possible to distribute this task to the agents and receive preprocessed and ready-to-store logs instead of parsing all logs at the central server. This approach can yield a significant reduction in CPU load on the central log server.

For more information about the patterns used by the NXLog agent, please refer to the pm_pattern module documentation in the NXLog Reference Manual.

94.1. Pattern Groups

Pattern groups are used to group patterns together which are used to match log messages generated by the same application or log source. Some pattern groups are not applicable to specific log sources. With pattern groups it is easy to exclude (or include) patterns which cannot match at all because the source would never generate such log messages. For example if there is no SSH service on a system, we should not try to match patterns in the SSHd group against the logs coming from this system.

Pattern groups also serve an optimization purpose. They can have an optional match criteria. One or more fields can be specified using either EXACT or REGEXP match. The log message is first checked against this match criteria. If it matches, only then will be the patterns belonging to the group matched against the log message.

To create a pattern group, the following form needs to be filled out.

Pattern Group

Properties

Name
sshd

Description
Patterns for SSH

Match

Field name	Type	Match	Value
SourceName	[STRING]	EXACT	sshd

Add Field **Delete**

Save **Help**

After form submission, the pattern group can be viewed:

Pattern Group

Properties

Name	ssh
Description	SSH server logs

Match

Field name	Match	Value
SourceName	EXACT	sshd

Patterns

Pattern Name	Pattern Description
ssh_auth_failure	ssh authentication failure
ssh_auth_success	Successful ssh authentication
ssh_pam_auth_failure	ssh pam authentication failure
ssh_pam_session_closed	ssh pam session closed
ssh_pam_session_opened	ssh pam session opened

Showing 1 to 5 of 5 entries

Permissions [+]

Delete **Edit** **Export** **Add pattern**

In the above example the ssh patterns will be only checked against the log if the field *SourceName* matches the string *sshd*. The *SourceName* field must be extracted from the Syslog message with a syslog parser prior to running the logs through the pattern matcher.

94.2. Creating a pattern

Patterns can be created directly by clicking on the **CREATE PATTERN** menu item. In this case an empty form must be filled out.

Create pattern

Pattern Info

Pattern Name SSH login event	Pattern Description This event is generated when a user is successfully authenticated over ssh
Pattern Group sshd	Create Group

Here you should enter the basic pattern information. Make sure the *Pattern Group* is set.

The next Match block must be populated with field(s) value(s). For example a message field:

Match

Field name	Type	Match	Value
Message	[STRING]	EXACT	Accepted public key for root from 192.168.255.98 port 59395 ssh2

Add Field **Update Test Cases**

This can be more generic according to our needs so that the pattern can extract the user name and the destination IP address from the message:

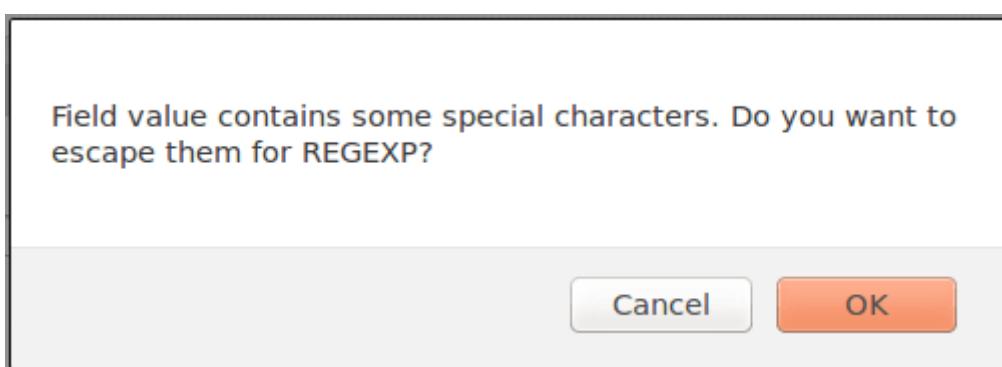
Match

Field name	Type	Match	Value
Message	[STRING]	REGEXP	^Accepted \S+ for (\S+) from (\S+) port \d+ ssh2
		Captured fields	Type
		AccountName	[STRING]
		DestinationIPv4Address	[IP4ADDR]

Add Field **Update Test Cases**

We replace those parts of the message with regular expressions constructs (`\S+` in the above example) which are not static. Captured substrings (the `(\S+)`) are stored in the fields we select. In the above example *AccountName* and *DestinationIPv4Address* are used to store the extracted values.

If it is necessary, add more than one field to execute the matching operation against. The match type can be either an *EXACT* or a *REGEXP* match. If this is toggled to *REGEXP*, the NXLog Manager will offer to escape special characters:



If the regular expression does not start with the caret (^), the regexp engine would try to find an occurrence anywhere in the subject string. This is a costly operation. Usually the provided regular expression is written to match the start of the string and it is easy to forget the caret from the start. For this reason the interface will show a hint:

Field name	Type	Match	Value	
Message	[STRING]	REGEXP	Accepted \S+ for (\S+) from (\S+) port \d+ ssh2	Delete consider using ^

NOTE

The regular expressions are compiled and executed by the NXLog engine using the PCRE library.
The regular expression must be [PCRE](#) compatible in order to work.

The last block is for optional testcases:

Test Case 0 X +

Field name	Type	Value	
Message	[STRING]	Accepted publickey for root from 192.168.255.98 port 59395 ssh2	Delete

[Add Field](#)

Captured fields

Field name	Type	Value	
AccountName	[STRING]	root	Delete
DestinationIPv4Address	[IP4ADDR]	192.168.255.98	Delete

[Add Field](#) [Calculate Fields](#)

This built-in testing interface is extremely useful for verifying the functionality of our patterns. Without this we could only find out that the pattern doesn't work if we loaded the pattern into the agent and ran it against a set of logs.

The field value was already filled in using the log message we used to create the pattern from. After clicking the Calculate Fields button, the captured field values should appear correctly. If they don't, you have made a mistake somewhere and should go fix it.

94.3. Message classification with patterns

Patterns can not only load a value from a captured substring into a field but additional fields may be specified. This feature can be used for message classification and to tag log messages with special values which can be later used in the processing chain.

Set

Field name	Type	Value	
TaxonomyAttack	[STRING]	AuthExploit	Delete
TaxonomyObject	[STRING]	Account	Delete
TaxonomyAction	[STRING]	Authenticate	Delete
TaxonomyProducer	[STRING]	OS	Delete
TaxonomyStatus	[STRING]	Success	Delete
VulnerabilityName	[STRING]	CVE-1234	Delete

[Add Field](#)

There are five special fields starting with *Taxonomy*. NXLog Manager comes with a dictionary for these taxonomy values and only a value from this dictionary list can be set. Event taxonomy is an important concept which allows to handle events generically regardless of their source.

If you do not want to classify the event with the Taxonomy fields or one or more is not applicable, click the Delete to remove it. The Taxonomy fields are optional but are recommended to be set. It is possible to add other additional fields and specify their value.

94.4. Searching patterns

The pattern list has a simple search input box in the upper right corner. This can search for entries in the list and will show rows which contain the specified keyword.

Patterns

	Pattern Name	Pattern Group	Pattern Description	Search: <input type="text"/>
<input type="checkbox"/>	chfn_change_user_info	chfn	Change user information	
<input type="checkbox"/>	kernel_boot_linux_version	kernel	Linux version info printed when booting	
<input type="checkbox"/>	login_pam_auth_failure	login	login pam authentication failure	
<input type="checkbox"/>	login_pam_session_closed	login	login pam session closed	
<input type="checkbox"/>	login_pam_session_opened	login	pam login session opened	
<input type="checkbox"/>	passwd_change	passwd	Password change	
<input type="checkbox"/>	shutdown_reboot	shutdown	Shutdown for system reboot	
<input type="checkbox"/>	ssh_auth_failure	ssh	ssh authentication failure	
<input type="checkbox"/>	ssh_auth_success	ssh	Successful ssh authentication	
<input type="checkbox"/>	ssh_multiple_auth_failure	login	ssh pam multiple authentication failures	
<input type="checkbox"/>	ssh_pam_auth_failure	ssh	ssh pam authentication failure	
<input type="checkbox"/>	ssh_pam_session_closed	ssh	ssh pam session closed	
<input type="checkbox"/>	ssh_pam_session_opened	ssh	ssh pam session opened	
<input type="checkbox"/>	useradd_create_account	useradd	New user creation on Linux	
<input type="checkbox"/>	userdel_delete_user	userdel	Delete user	

Showing 1 to 15 of 15 entries

◀ ▶ 1 ▷ □

[Create](#) [Delete](#) [Export](#) [Export All](#) [Import](#) [Edit](#) [View](#) [Help](#)

There is a more powerful search interface which allows searching in any of the patterns' properties (fields, test cases, etc). Click on the [SEARCH PATTERN](#) menu item under the PATTERN menu.

Search Pattern

Field restriction

▾

Search criteria

[Search](#) [Help](#)

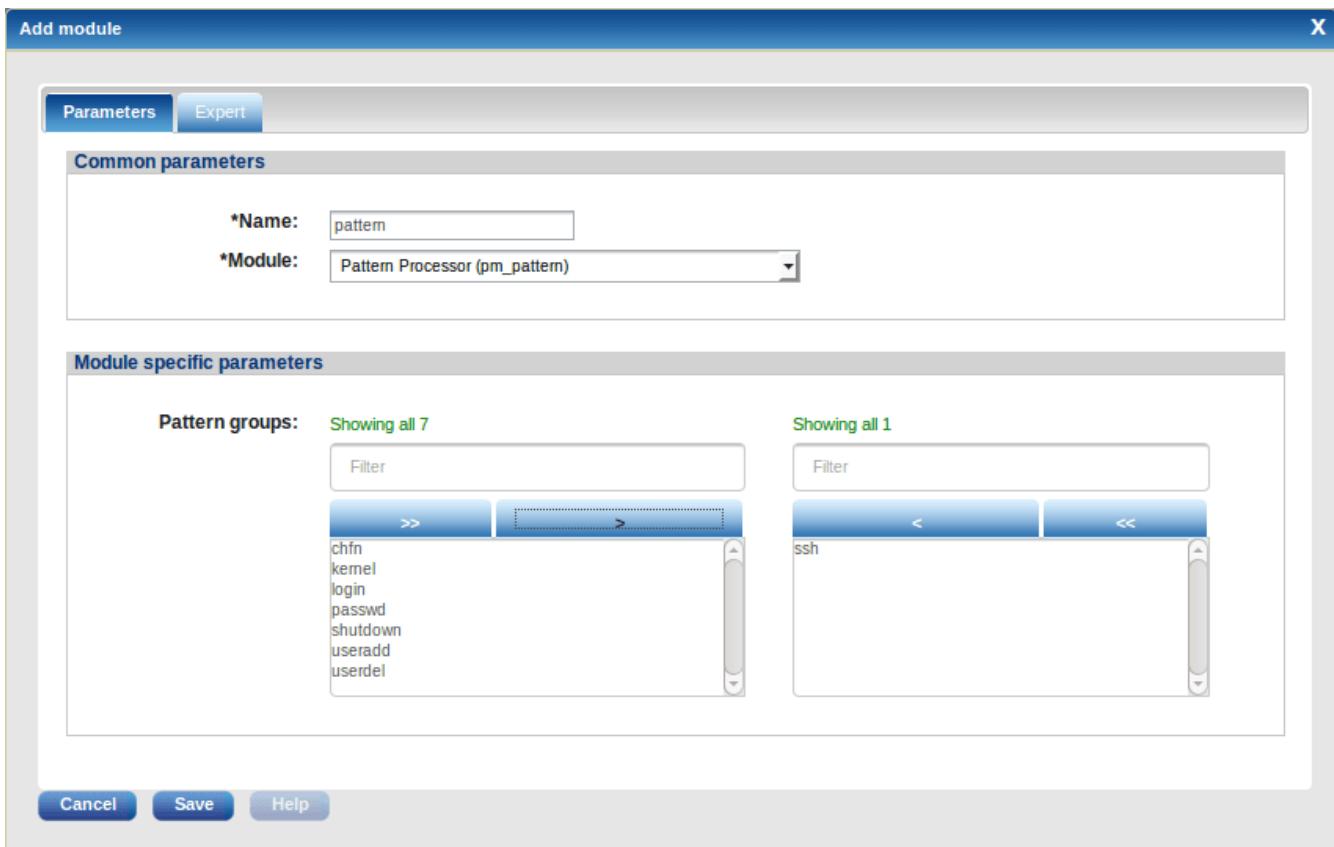
94.5. Exporting and importing patterns

NXLog Manager can export and import patterns in an XML format. This is the same format used by the NXLog agent. To export a pattern or a pattern group, check its checkbox in the list and click Export. You can import a pattern database file by clicking on the **IMPORT PATTERN** menu item or the Import button under the pattern list.

94.6. Using patterns

Patterns are used and executed by the NXLog engine. Unlike other log analysis solutions which utilize a single pattern matcher in the central engine, the architecture of NXLog Manager allows patterns to be used on the agents as well.

To use the patterns in an NXLog agent, add a *pm_pattern* processor module and select the appropriate pattern groups:



The patterns will be pushed to the NXLog agent after clicking Update config and they will take effect after a restart. See the [Agents](#) chapter for more information about agent configuration details.

NOTE

Some patterns work with a set of fields and this requires some preprocessing (e.g. syslog parsing) in some cases. Instead of writing a regular expression to match a full Syslog line which includes the header (priority, timestamp, hostname etc), it is a lot more efficient to write the regular expression to match the Message field (instead of the raw_event field) and have a syslog parser store the header information in separate fields before the pattern matching. These patterns will be usable when the same message is collected over a different protocol.

Chapter 95. Correlation

Event correlation is an important concept in log analysis. Each log message contains information about an event which occurred at some point. There are cases when the occurrence (or absence) of one or more events must be treated specially.

A trivial example for this is to detect 3 failed login attempts into a system. When this happens, the user will be likely locked out. If the log analysis system is capable to detect such situation, a lot of things can be automated. You will not need to wait for the user to come asking for a new password.

Event correlation in NXLog Manager is architected similarly to the [Pattern](#) system. It is performed in real-time by the NXLog agents. This way it is possible to do local event correlation on the client side (at the agent). This will not only reduce load on the central server but the system can send alerts over another channel (e.g. SMS) even if the network is down and the log messages would not reach the central log server.

For more information about the correlation capabilities of NXLog Manager, please consult the NXLOG Reference Manual and see the documentation about the pm_evcorr module.

95.1. Correlation rulesets

Correlation rules are grouped into rulesets. Because there can be different correlation rules depending on the agent or log source, rulesets ease the use of correlation rules. To view the list of correlation rulesets, click the **CORRELATION** menu item. The list of existing correlation rulesets will appear such as in the following screenshot:

Rulesets

	Name	Count
<input type="checkbox"/>	main-ruleset	2

To create a new ruleset, click the **[Add]** button.

Add ruleset X

*Name

Cancel Save Help

95.2. Correlation rules

Correlation rules check whether the conditions specified in the rule are satisfied and execute an action. Correlation rules are evaluated linearly.

Clicking the name of a correlation ruleset will show a list of correlation rules within the ruleset:

main-ruleset

The screenshot shows a web-based configuration interface for a ruleset named "main-ruleset". At the top, there are three tabs: "Edit" (selected), "Permissions", and "Tags". Below the tabs is a search bar labeled "Search:" with a placeholder box. The main area displays a table of rules:

<input type="checkbox"/>	Name	Type
<input type="checkbox"/>	ssh bruteforce	Thresholded
<input type="checkbox"/>	simple	Simple

At the bottom of the table are navigation buttons: "Add", "Delete", "Up", "Down", "Export", and "Help". To the right of the table are page navigation icons: back, forward, and a page number "1".

NOTE

The order of the rules within the ruleset matter because they are evaluated by NXLog's pm_evcrr module in the order they appear. To change the order of the rules, use the Up and Down buttons.

95.2.1. Creating correlation rules

To create a new correlation rule, click the Add button on the bottom of the list. The following dialog window will appear:

The "Add rule" dialog window has the following fields:

- *Name:** A text input field.
- *Type:** A dropdown menu set to "Simple".
- *Condition:** A dropdown menu set to "matched pattern is:" followed by a "select pattern" button.
- Action:** A large text input field.
- Verify:** A blue "Verify" button below the action field.

At the bottom of the dialog are "Cancel", "Save", and "Help" buttons.

Each correlation rule has a mandatory *Name*, *Type* and *Action* parameter and one or more type specific parameters where the conditions can be specified. The following correlation rule types are available:

- Simple
- Suppressed
- Pair
- Absence
- Thresholded

Please consult the NXLog Reference Manual and see the documentation about these rule types provided by the pm_evcrr module. There are two modes available to specify a condition

Matched pattern is

This will generate a simple test to check whether the specified pattern matched. The generated NXLog config will contain a similar snippet:

```
if $PatternID == 42 {\  
    ACTION  
}
```

NOTE

For this to work, the pm_pattern module must be configured and must be in front of the pm_evcorr module in the route. The pm_pattern module is responsible for setting the PatternID field.

Expert

The Expert field expects a statement (a boolean condition) which evaluates to TRUE or FALSE. The above expressed in Expert form would look like the following:

```
$PatternID == 42
```

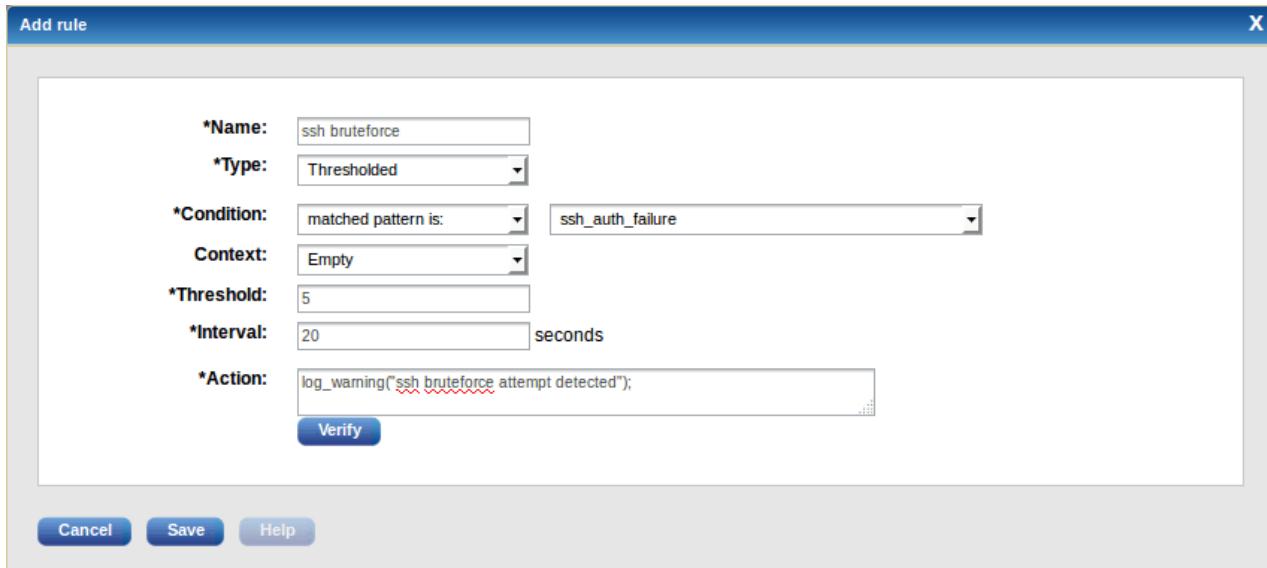
Using the language constructs provided by NXLog, it is possible to specify more complex conditions here, e.g.

```
($EventTime > now() - 10000) and ($PatternID == 42 or $PatternID == 142)
```

Example 510. Correlation rule for ssh bruteforce attack detection

In this example we will create a correlation rule which will detect ssh bruteforce attempts. We define a bruteforce attempt as 5 login failures within a 20 second interval. In this example only an internal warning message is generated, but it is possible to trigger any other action, for example execute an external script to block the IP or send an alert in email.

This correlation rule depends on a pattern which matches an ssh authentication failure event. See the [Creating patterns](#) section on how to do this. Once the pattern is available in our database, the correlation rule should be configured as shown on the following screenshot:



95.3. Exporting and importing correlation rules

NXLog Manager can export and import correlation rules and rulesets in an XML format. To export a correlation rule or a ruleset, check its checkbox in the list and click Export. You can import a correlation rule XML file by clicking on the Import button under the ruleset list.

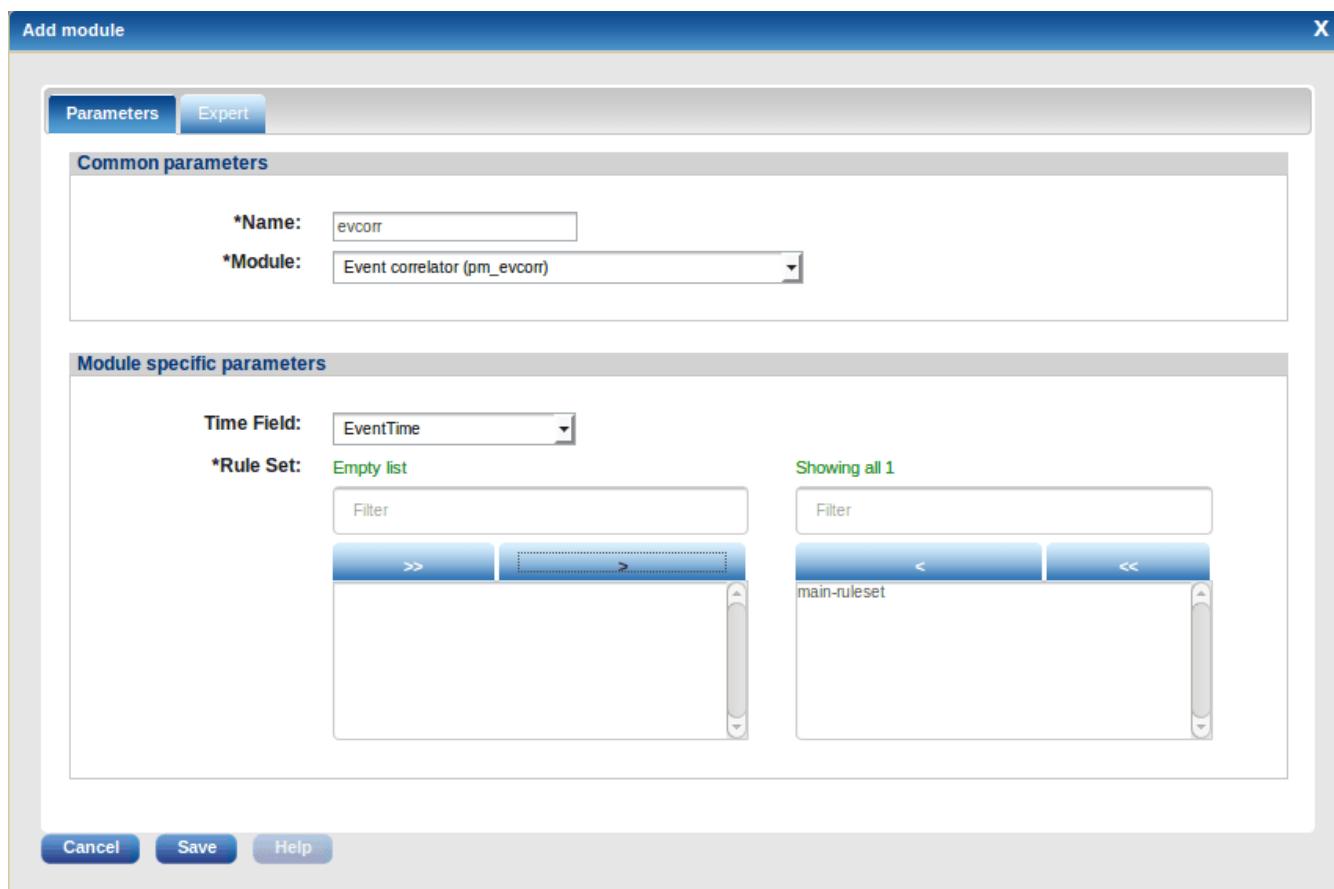
NOTE

Unlike patterns, correlation rules used by NXLog are not in XML. Correlation rules exported from NXLog Manager cannot be used by NXLog because the NXLog agent uses apache-style configuration file for the rules and this is part of (or included in) the pm_evcrr module configuration block in `nxlog.conf`.

95.4. Using correlation rules

Similarly to patterns, correlation rules are also used and executed by the NXLog engine. Unlike other log analysis solutions which utilize a single pattern correlation engine in the central log server, the architecture of NXLog Manager allows correlation rules to be evaluated on the agents as well.

To use the correlation rules in an NXLog agent, add a `pm_evcrr` processor module and select the appropriate correlation ruleset:



It is recommended to use only one ruleset per agent. The correlation rules will be pushed to the NXLog agent after clicking Update config and they will take effect after a restart. See the [Agents](#) chapter for more information about agent configuration details.

NOTE

In many cases correlation rules depend on patterns (and the `PatternID` field). For this reason a `pm_pattern` module should be in the processor chain before the `pm_evcrr` module.

Chapter 96. Agents

NXLog agents are used to collect and store event log data. This chapter discusses the GUI configuration and management frontend provided by NXLog Manager. For more information about the NXLog agent please refer to the NXLOG Enterprise Edition Reference Manual.

The NXLog instances can be managed, monitored and configured remotely over a secure management channel. The management component in NXLog Manager is called the agent manager. There are two operation modes:

- The NXLog agent initiates the connection and connects to the agent manager
- The agent manager initiates the connection and connects to the NXLog agent

Mutual X509 certificate based authentication is used over a trusted secure TLS/SSL channel in order to guarantee that only an authorized NXLog agent can connect and only an authorized entity can manage the agent. The agent manager queries the status information of each NXLog agent every 60 seconds.

To list the available NXLog agents, click on the **AGENTS** menu. A list similar to the following should appear:

	Status	Agent name	Version	Host	Started	Load	Mem. usage	Received	Received today	Processing	Sent	Sent today
<input type="checkbox"/>	●	debian-m4a88t-m	3.0.1706	192.168.0.109	2016-10-07 13:37	1.74	<div style="width: 8.48M;">8.48M</div>	<div style="width: 3px;">3</div>	<div style="width: 230px;">230</div>	<div style="width: 0px;">0</div>	<div style="width: 3px;">3</div>	<div style="width: 230px;">230</div>
<input type="checkbox"/>	○	test	Unknown									
<input type="checkbox"/>	○	WIN200JP	Unknown									
<input type="checkbox"/>	○	win2003	Unknown									
<input type="checkbox"/>	○	win2008-builder	Unknown									

Showing 1 to 5 of 5 entries

Refresh status Start Stop Update and reload Configure Add Delete Clone Download config View log Assign template Issue certificate Renew certificate Help

This list contains the following information:

Status

There is coloured ball showing agent's status. It can be any of the following:

Green - Online

If the last status response was successfull and the agent is functioning normally, the status is *Online*.

Grey - Offline

When there is no connection with the agent manager the status is *Offline*. Usually this indicates that there is a network error, the agent is stopped, has not been started or is misconfigured. You should check the NXLog agent's *LogFile* for error diagnostics.

Red - Error

This status is shown when one or more modules aren't running. For network output modules this can be shown when there is no connection and the module is unable to send. This can be fixed by starting the module or solving the problem which prevents the module to function correctly. You should check the NXLog agent's *LogFile* for error diagnostics.

Yellow - Unmanaged

When the agent is configured to be **Unmanaged**, it is not possible to administer it remotely.

Yellow - Untrusted

When the agent is connected to the manager without own certificate. The manager accepts it, but with warning. For those agents must be issued certificates if they are freshly installed without certificate.

Yellow - Forged

When the agent is connected to the manager with certificate which CN doesn't match the reverse DNS of this agent. Most probably it's certificate is copied/configured from another agent, thus for it must be issued certificate as well.

If the agent configuration has been updated either centrally (by the application), or locally (on the agent side), there will be small yellow warning sign next to the ball with tooltip of the message. You must issue [Update config](#) command in order to apply the central configuration. If the configuration has been changed locally, a confirmation will be asked.

Agent name

The agent name is taken from the certificate subject name. For this reason the same name must be used here as the [certificate subject](#). Clicking on the agent name will load the [Agent information](#) page.

Template

The assigned template to this agent. Agents inherit all the template configuration. For more information about templates, see [Templates](#) chapter.

NOTE

This column is not visible by default. To enable it, select it from the column list by clicking on the round *Configuration* button on the top left of the table.

Tags

The assigned tags to this agent (usually they are assigned to its template). For more information about configuring tags for templates, see [Tags](#) section.

NOTE

This column is not visible by default. To enable it, select it from the column list by clicking on the round *Configuration* button on the top left of the table.

Version

This is the NXLog agent version number.

Host

This is the IP address of the remote socket where the NXLog agent has connected from. Not available when the agent is *Offline* or *Unmanaged*.

Started

Shows the time when the NXLog agent has been started. Not available when the agent is *Offline* or *Unmanaged*. The value is set when the NXLog service is started or restarted. This is not reset when using the [Reload](#) button.

Load

The system load as reported by the operating system where the NXLog agent is running. This is not implemented on some platforms (notably Microsoft Windows), in this case *Unknown* will be shown.

NOTE

This value represents the system load of the operating system and not that of the NXLog agent. Due to other resource intensive processes this can be high even if the NXLog agent is idle.

NOTE

There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

Mem. usage

The amount of memory used by the NXLog agent. On some platforms *Unknown* is shown where this information is not available.

NOTE There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

Received

The sum of log messages received by all input modules since the agent has been started.

NOTE There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

Received today

The sum of log messages received by all input modules for the last 24 hours.

NOTE There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

Processing

Each NXLog agent module has a separate queue. This number shows the sum of messages in all modules' queues.

NOTE There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

Sent

The sum of log messages written or sent by all output modules since the agent has been started.

NOTE If you have two output modules writing/sending logs from a single input, the number under *Sent* will be double of the value under *Received* for obvious reasons.

NOTE There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

Sent today

The sum of log messages written or sent by all output modules for the last 24 hours.

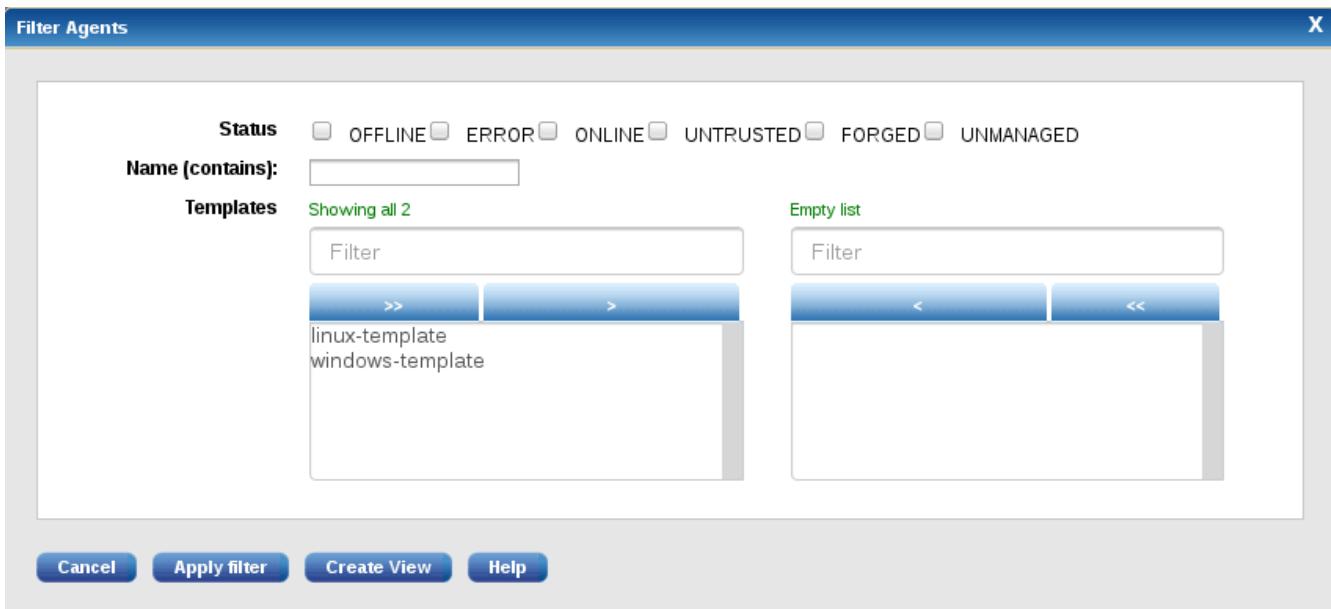
NOTE If you have two output modules writing/sending logs from a single input, the number under *Sent* will be double of the value under *Received* for obvious reasons.

NOTE There is small thumbnail graph for the statistics of last 10 average values.

This information is not available when the agent is *Offline* or *Unmanaged*.

The information shown in the agent list is refreshed every 60 seconds or when **Refresh status** is clicked.

On the top of the list there is a **[Filter agents]** button which can be used for agent filtering on that list. When clicked, the following dialog appears:



Here agents can be filtered by 3 criteria: Status(es), Agent name (contains the character sequence specified), and Template(s). Clicking [**Apply filter**] will refresh the agent list with only agents matching filtering criteria. For example selecting ONLINE status will show the following:

When a filter is applied, next to Filter agents appears another button [**Clear filter**] which can be used to discard the applied filter and show the list with all agents again.

On the Filter Agents dialog there is also another option to save this filter in the configuration database as an Agents View. Clicking on [**Create View**] button will popup another dialog to enter the view name:

The view name must be unique and not containing any special characters and blank spaces. All saved views will appear as tabs next to Agent templates tab and the newly created view will be immediately selected automatically:

On the bottom of the list there is a row of buttons which can be used to manage the agent(s).

Refresh status

This will send a query to the agent to retrieve status information. You need to select at least one *Online* agent with the checkbox to be able to use this.

Start

Start all stopped modules. You need to select at least one *Online* agent with the checkbox to be able to use this.

NOTE

The xm_soapadmin module responsible for the agent manager connection must be always running, so this module is not affected. The NXLog process cannot be stopped and/or started from the NXLog Manager management interface.

Stop

Stop all modules. You need to select at least one *Online* agent with the checkbox to be able to use this.

NOTE

The xm_soapadmin module responsible for the agent manager connection must be always running, so this module is not affected. The NXLog process cannot be stopped and/or started from the NXLog Manager management interface.

Update config

After the agent settings were changed, use this button to push the new configuration to the agent. All configuration related files, including pattern database files and certificates will be pushed to the agent. You will most probably want to **Reload** the agent after updating the configuration in order for the new settings to take effect. You need to select at least one *Online* agent with the checkbox to be able to use this.

Reload

This forces the agent to stop and shutdown all modules, reload the configuration and then reinitialize and start all. This should be used after a new configuration was pushed to the agent in order for the new setting to take effect. You need to select at least one *Online* agent with the checkbox to be able to use this.

NOTE

This is not a process/service level restart but rather a reload. The xm_soapadmin module responsible for the agent manager connection must be always running, so this module is not affected. The NXLog process cannot be stopped and/or started from the NXLog Manager management interface.

Configure

This button will load the [Agent configuration](#) page. You need to select exactly one agent with the checkbox to be able to use this.

Add

Add a new agent.

NOTE

An agent will appear in the list without a configuration after successfully connecting to the agent manager even if it does not exist in the agent list. It is possible to add a new agent by creating a certificate, deploying the installer and starting the NXLog service. The new agent entry should appear automatically.

Delete

Delete the agent.

NOTE

There is no confirmation dialog, be careful when clicking on this button.

NOTE

The agent will reappear if it has a valid certificate and can successfully authenticate to the agent manager. Make sure to revoke the certificate and stop the NXLog service before you delete the entry with this button. If the NXLog service is not stopped and removed, it will try to reconnect (if it was configured so).

Clone

Clone the agent. The cloned agent(s) will have all the modules and routes of the original one. You need to select exactly one agent with the checkbox to be able to use this.

Download config

Downloads the agent configuration in zip file to ease agents deployment locally. For each agent there will be folder in the archive with its name containing all the necessary configuration files and certificates. You must select at least one agent with the checkbox to be able to use this.

View log

View the log of an agent. By the default it is limited to 100K of the last log. You need to select exactly one agent with the checkbox to be able to use this.

Assign template

Assign template to agent(s). The selected agents will lose their configuration and now will have the template one.

NOTE

This button is only visible if there are existing templates.

You must select at least one agent with the checkbox to be able to use this.

Issue certificate

Issue certificate for the selected agent(s). If the checkbox *Update connected agents* remains checked, the manager will also issue [Update config](#) command. You must select at least one agent which doesn't have certificate already with the checkbox to be able to use this.

Renew certificate

Renew certificate for the selected agent(s) due to expiry or other reason. If the checkbox *Update connected agents* remains checked, the manager will also issue [Update config](#) command.

NOTE

If a selected agent has already valid certificate, it will be revoked by the system.

You must select at least one agent with the checkbox to be able to use this.

96.1. Agent information

The following agent information page is loaded when an *Online* agent is clicked.

Started	2015-06-04 18:21	Agent name	192.168.56.102
Version	2.8.1249	Load	0.26
Host	192.168.56.102	PID	2737
Mem. usage	3.3M	Hostname	debian-vpc
OS	Linux	Agent time	2015-06-04 18:25:11
System info OS: Linux, Hostname: debian-vpc, Release: 3.2.0-4-amd64, Version: #1 SMP Debian 3.2.68-1+deb7u1, Arch: x86_64, 1 CPU(s), 1002.9Mb memory			
Download links CA Certificate Key			
Refresh status Start Stop Update config Reload Delete Clone View log Help			

The page will show less information if the agent is not connected to the agent manager. The action buttons on

this page are not discussed as they are the same as on the agent list described previously.

If the agent is *Online* and some of its modules has variables or statistical counters, they will appear on this page in a table.

Clicking on the Modules tab will show detailed information in a table about each module as shown in the following image.

	Name	Module	Type	Status	Received	Processing	Sent	Dropped
<input type="checkbox"/>	db	om_null	OUTPUT	RUNNING	0	0	0	0
<input type="checkbox"/>	exec	im_file	INPUT	RUNNING	0	0	0	0

Buttons at the bottom: Refresh status, Start, Stop, Restart, Help.

This information is only available when the agent is Online and contains the following information.

Name

The name of the module instance.

Module

The type of the loadable module which was used to create the module instance.

Type

The type of the module. It can take the following values:

- INPUT
- PROCESSOR
- OUTPUT

Extension module instances are not shown.

Status

- STOPPED
- RUNNING
- PAUSED
- UNINITIALIZED

NOTE

The module may become PAUSED if it cannot send or forward the output. This is caused by the built-in flow control and is perfectly normal unless you see the module in this status for a longer period and the number of sent messages does not increase. You do not need to start the module when it is PAUSED, it will resume operations automatically.

Received

The number of log messages received or read.

Processing

The number of log messages in the module's queue waiting to be processed.

Sent

The number of log messages written or sent by the module.

Dropped

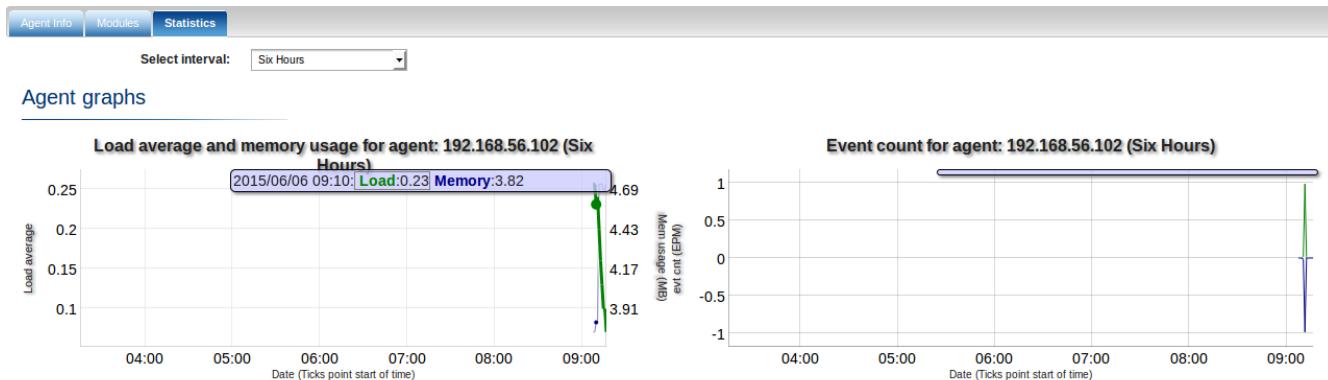
The module may filter and drop some messages. This number will be shown here. This is calculated using the value reported in Received and Sent.

Clicking on the Statistics tab will show several fully interactive graphs. There will be a graph for the following parameters:

- System load & Memory usage
- Total event count
- Event counts for each module
- Optionally can be added graphs for module variables and statistical counters by clicking the Add chart button. There must be selected the module and filled the name of the variable. Regular expressions of the name are also supported.

The interval of the graph can be selected using the combo box to be one of the following:

- Six hours
- One day
- One week
- One year



96.2. Agent configuration

Click on the [Configure](#) button or the Configuration tab to load the configuration form. The following global configuration tab should appear.

The screenshot shows the NXLog Agent configuration interface. The 'Configure' tab is selected. The 'Global' tab is active. The configuration page includes the following fields:

- *Agent connection type:** Connect to agent manager
- *Address:** 192.168.56.1
- *Port:** 4041
- *Certificate:** central-collector
- *Log level:** INFO
- Log to file:**
- Verbatim config:** [Empty text area]

At the bottom are 'Save' and 'Help' buttons.

The list of parameters are explained below.

Agent name

Set this to the certificate subject name. It is automatically filled out when the agent has connected and was automatically added.

Connection type

Unmanaged

Set the connection type to Unmanaged if you do not want to administer the agent remotely over a secure connection.

Listen (accept agent manager connection)

The NXLog agent will listen on the IP address and port for incoming SSL connections. You must also [configure the agent manager](#) to initiate connection to the agents.

Connect to agent manager

The NXLog agent will initiate the connection to the agent manager.

Address

The address where the agent should connect to or where the agent is listening on, depending on how the [Connection type](#) is configured.

Port

The port number where the agent should connect to or where the agent is listening on, depending on how the [Connection type](#) is configured.

Certificate

The certificate which will be presented by the NXLog agent during the mutual authentication phase when the connection is established with the agent manager. The agent manager will check whether the agent certificate has been signed with the [CA configured on the Agent Manager settings tab](#).

Loglevel

The loglevel to use when sending internal messages to the logfile and the im_internal input module.

Log to file

Enable this if you want to use a local `nxlog.log` file where the internal events of the NXLog agent are written. This method is more efficient and error resistant than using im_internal and it also works with DEBUG loglevel.

Verbatim config

Verbatim configuration text for this agent. This configuration will be put in the log4ensics.conf file as it is.

The list of modules can be managed independently regardless of the route they belong to. The following screenshot shows an example list of modules.

The screenshot shows the NXLog UI interface. At the top, there are two tabs: "Info" and "Configure". Below them is a navigation bar with five tabs: "Global", "Modules" (which is selected and highlighted in blue), "Routes", "Config", and "Permissions". On the right side of the interface, there is a search bar labeled "Search:" followed by a text input field. The main area displays a table of modules. The columns are "Name", "Module", and "Type". The rows contain the following data:

Name	Module	Type
db	om_null	OUTPUT
evcorr	pm_evcorr	PROCESSOR
internal	im_internal	INPUT
sslin	im_ssl	INPUT
sslout	om_ssl	OUTPUT

At the bottom of the module list, there are four buttons: "Add", "Delete", "Copy", and "Help".

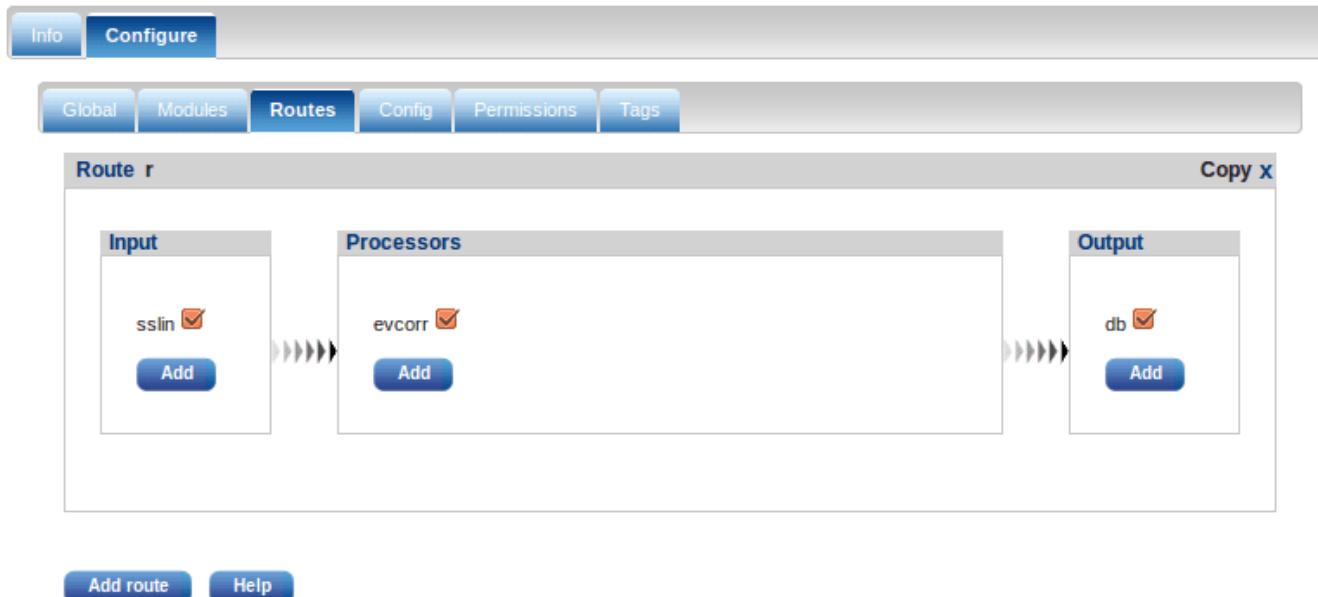
Click the [Add] button to add a new module. The module configuration dialog will pop up. To remove a module, click the checkbox after the module's name. Modules which are already part of a route cannot be removed. Go to the Routes tab to remove or add modules to a route. On the other hand modules not part of a route can only be removed on this list. Configuration will not be generated for modules which are not part of a route. Click on copy to copy this module configuration to other agents. A popup will appear to select them. Click on the module's name to modify its configuration.

To configure the flow of log data in the NXLog agent, click the Routes tab. A freshly created agent does not have any routes. Click the Add route button to add a route.

The screenshot shows the "Add route" dialog box. It has fields for "Name" and "Priority". The "Name" field is a text input box, and the "Priority" field is a dropdown menu set to "Default". At the bottom of the dialog are two buttons: "Cancel" and "Save".

Enter the name and select the priority. Data will be processed in priority order among routes. Lower gets processed first. This is only useful if you have more routes which contain different input sources. Select default if you do not wish to assign a priority value.

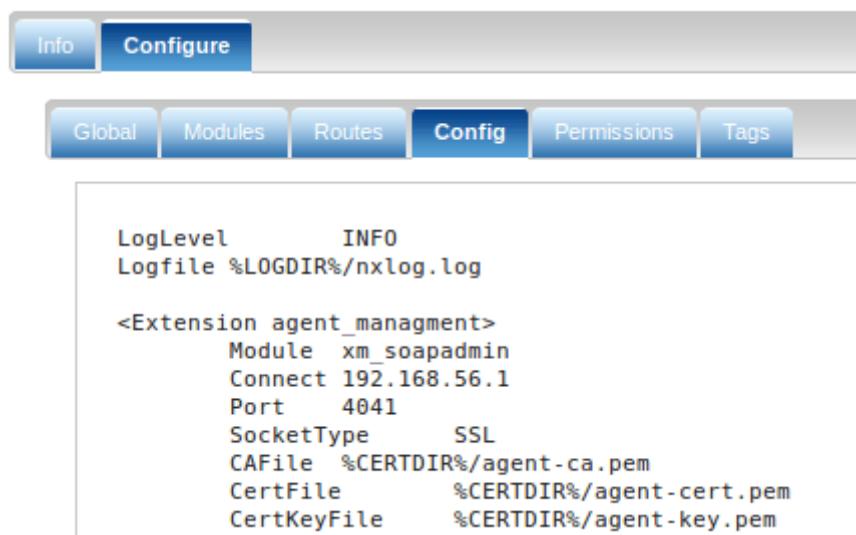
After the route is added, you can now add modules to it. A route requires at least one input and one output module. The following screenshot shows an example of a route with one module for each type.



Click the **[Add]** button inside the input/processor/output block to add a module instance. The [module configuration dialog](#) will pop up. If there is already an existing module instance, you will be able to select that also. It is possible to add more module instances to each block. To remove a module, uncheck the checkbox after its name. The module instance is only removed from the route. To fully delete it, click the Modules tab and remove the module.

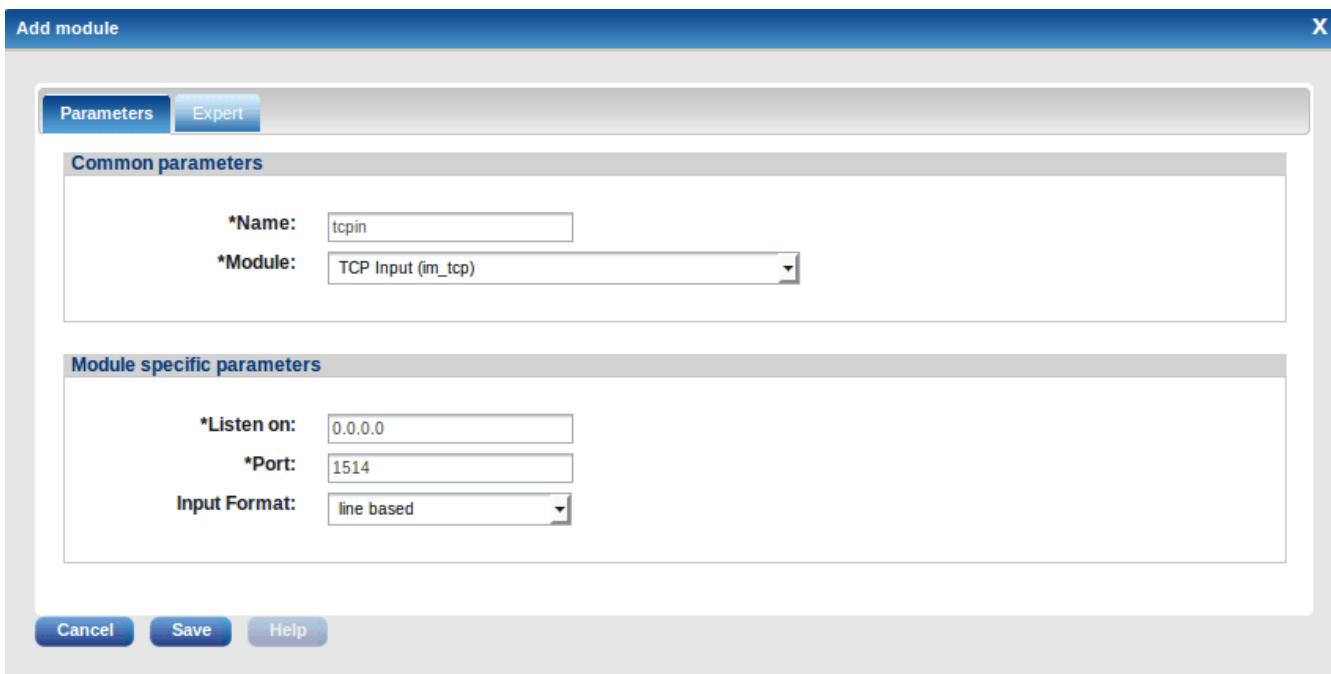
As with modules, there is option to copy entire route to other agents. Click on Copy link on top right of a route to select one or more agents to copy to.

The last tab contains the generated nxlog configuration which will be pushed to the NXLog agent when clicking on [Update config](#) as shown in the following screenshot.



96.2.1. Module configuration

When a new module instance is created, the following dialog window is shown.



The module configuration dialog *Parameters* tab consists of two blocks: Common parameters and Module specific parameters. The Common parameters are as follows:

Name

The name of the module instance.

Module

The loadable module which is used to create the module instance.

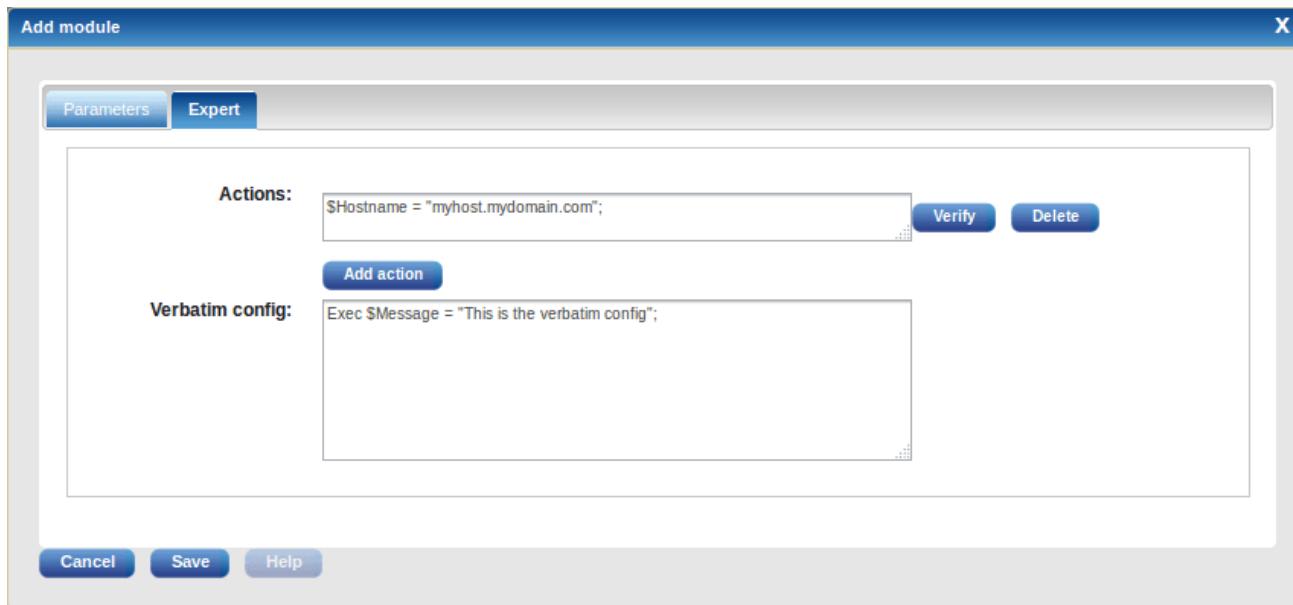
The module configuration dialog *Expert* tab consists of:

Actions

The Action text area can be used to input statements in NXLog's configuration language. It is possible to add multiple Action input widgets. Add each action with the Add action button. Click Verify to verify the statement(s). The contents of the Action block are copied into the module's Exec directive. Newline characters will be replaced with the backslash escape character. The statement entered on the above screenshot is highlighted below in the generated NXLog configuration.

```
<Input sslin>
    Module  im_ssl
    Host    localhost
    Port    5432
    InputType      Binary
    CAFile   %CERTDIR%/sslin-ca.pem
    CertFile  %CERTDIR%/sslin-cert.pem
    CertKeyFile  %CERTDIR%/sslin-key.pem
    AllowUntrusted FALSE
    RequireCert  TRUE
    Exec    $Hostname = "myhost.mydomain.com";
    Exec    $Message = "This is the verbatim config";
</Input>
```

Verbatim config



The following is generated into `nxlog.conf` from the above:

```
<Input sslin>
    Module  im_ssl
    Host    localhost
    Port    5432
    InputType      Binary
    CAFile   %CERTDIR%/sslin-ca.pem
    CertFile    %CERTDIR%/sslin-cert.pem
    CertKeyFile  %CERTDIR%/sslin-key.pem
    AllowUntrusted FALSE
    RequireCert   TRUE
    Exec  $Hostname = "myhost.mydomain.com";
    Exec $Message = "This is the verbatim config";
</Input>
```

Module specific parameters are not discussed in this user manual. Please consult the NXLog Enterprise Edition Reference Manual for more information about each module and their capabilities.

Chapter 97. Templates

NXLog templates are used for [Agent configuration](#) with templates. Templates can be useful for agent mass deployment with the configuration of the template, and also for tagging. This chapter discusses the GUI configuration and management frontend provided by NXLog Manager.

To list the available NXLog templates, click on the Agent templates tab in [AGENTS](#) page. A list similar to the following should appear:

The screenshot shows a table with the following data:

Template name	Description	Connection address	Connection port	Created	Last modified
linux-template	Template for linux agents	192.168.0.109	4041	2016-10-06 16:19:37.0	2016-10-06 16:19:37.0
windows-template	Template for windows agents	192.168.0.109	4041	2016-10-06 16:19:56.0	2016-10-06 16:19:56.0

Showing 1 to 2 of 2 entries

Buttons at the bottom: Add, Delete, Clone, Create agents, Help

This list contains the following information:

Template name

The template name is the unique name of a NXLog template. Clicking on the template name will load the [Template configuration](#) page.

Description

This is the NXLog template detailed description.

Connection address

This column shows the address of the NXLog agents belonging to this template are configured for connection with the [NXLog Manager](#). Not available when the template is Unmanaged.

Connection port

This column shows the port of the NXLog agents belonging to this template are configured for connection with the [NXLog Manager](#). Not available when the template is Unmanaged.

Created

This column shows the date and time when this NXLog template is created.

Last modified

This column shows the date and time when this NXLog template has been last modified.

On the bottom of the list there is a row of buttons which can be used to manage the agent(s).

Add

Add a new template.

Delete

Delete the template.

Deleted template must be unassigned from the agents belonging to it. If there are any, a confirmation dialog will appear:

NOTE

**Are you sure you want to delete: linux-template?
The template is assigned to one or more agents.**

Cancel**OK***Clone*

Clone a template with the same configuration.

Create agents

Create agents and automatically assign this template to them.

97.1. Template configuration

As templates are used for grouping the NXLog agents configuration, this is almost the same as the [Agent configuration](#). The only difference is the missing [Certificate](#) as this is specific to the agents themselves.

97.1.1. Tags

NXLog templates/agents can be managed by tags. Tags have role and user access permissions. To list them for a template, click on Tags tab under Template configuration page:

Template name: linux-template

Name	Description	Permissions by Role	Permissions by User
Linux	tag for Linux templates	ROLE_ADMINISTRATOR: Administration ROLE_AGENT: Read Only	admin: By role John: Administration
SiteA	tag for SiteA templates	ROLE_ADMINISTRATOR: Administration ROLE_AGENT: Read Only	admin: By role John: Administration

Add **Edit** **Assign**

This list contains the following information:

Name

The unique name of a tag.

Description

This is the tag detailed description.

Permissions by Role

This column shows the access permissions of each role which is allowed to manage NXLog agents.

Permissions by User

This column shows the access permissions of each user which is allowed to manage NXLog agents.

On the bottom of the list there is a row of buttons which can be used to manage the tags(s).

Add

Add a new tag.

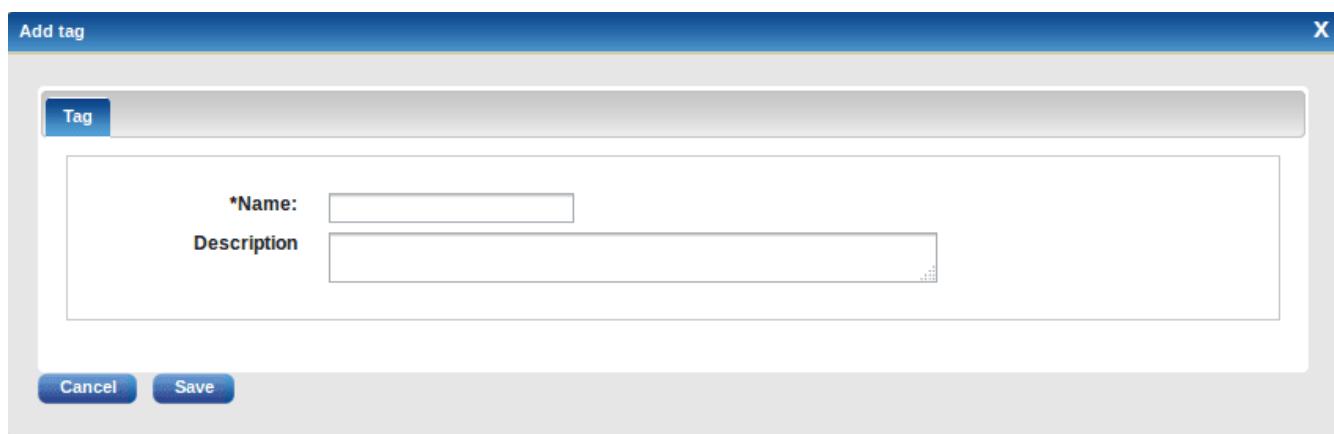
Edit

Edit a tag.

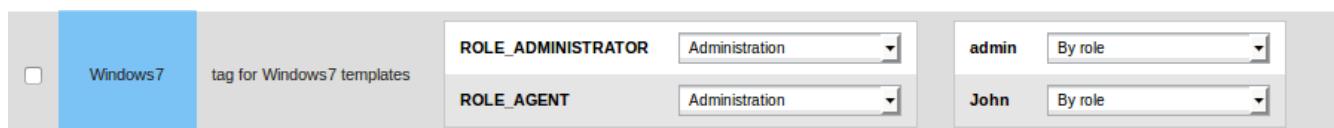
Assign

Assign (or unassign) existing tags to this template.

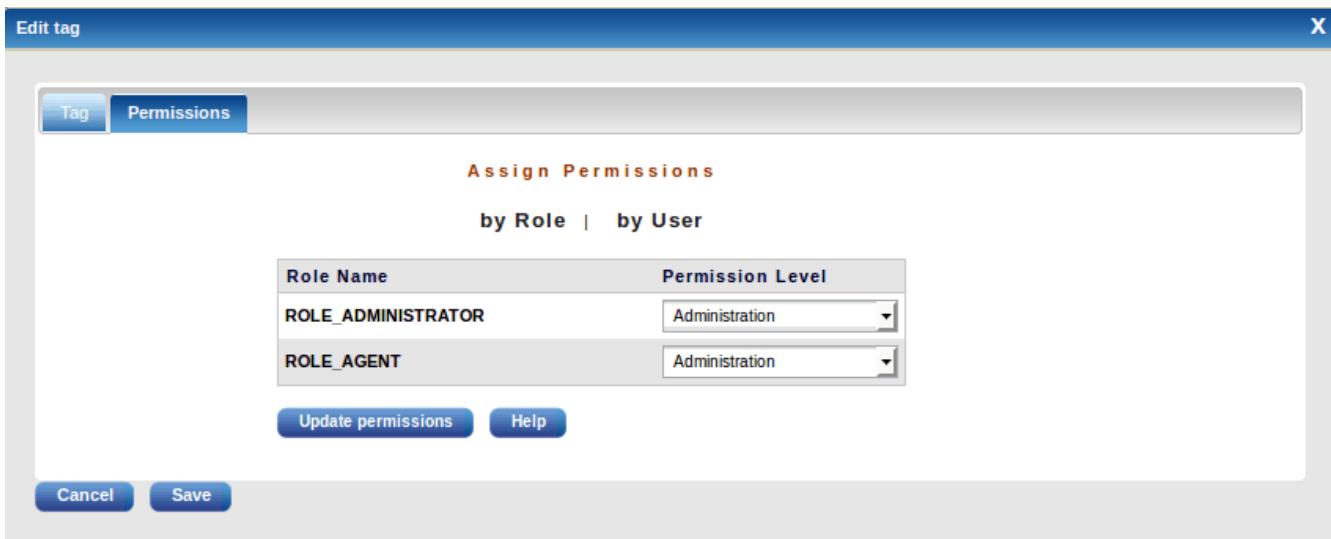
To add a new tag to the system (and in parallel assign to this template), click on the **Add** button. An Add tag dialog will appear:



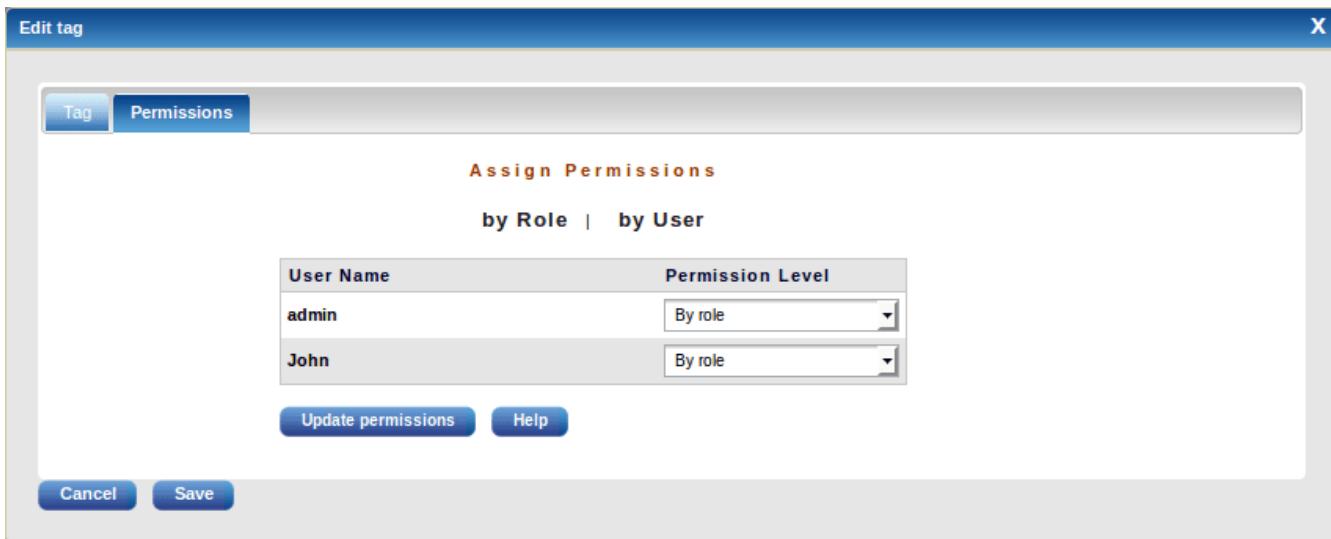
Fill in the Name and optionally description of this tag. When click on Save a new tag will be created with default access permissions, assigned to this template and will appear on the list:



Tag can be then edited by selecting it and click the Edit button. Edit dialog appears, containing two tabs: Tag and Permissions:

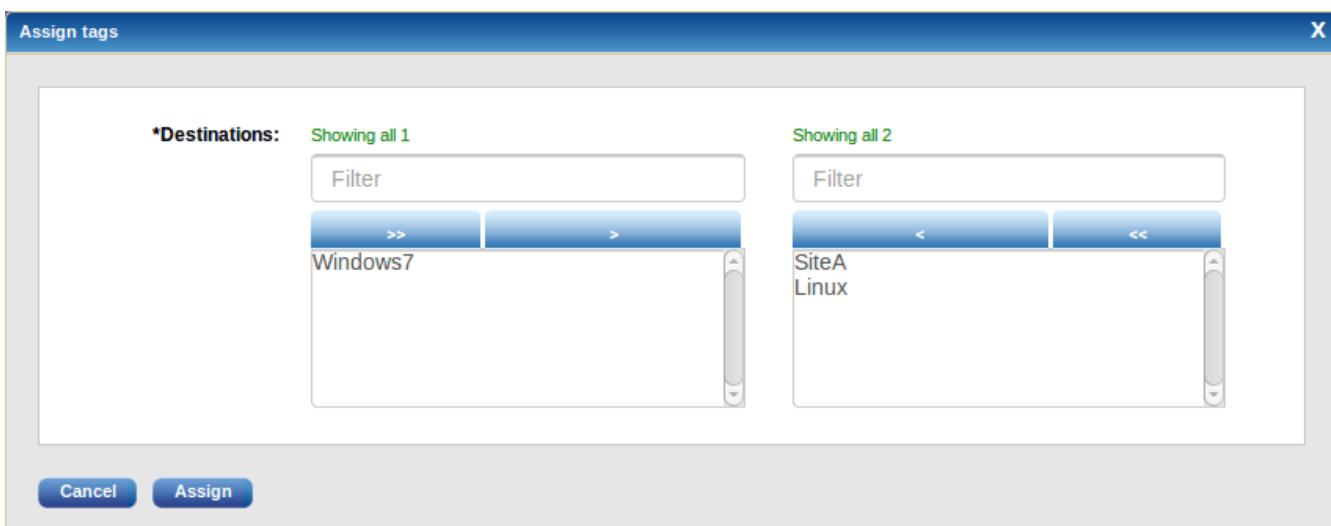


If permission need to be changed by user, click on by User link:



When some permissions are changed, click Update permissions button. When all done Save, to make sure all the changes will be accepted.

If tags are needed to be assigned/unassigned to the current template, click on Assign button on tag list page. The following dialog will appear:



Select the tags needed from the multi select box and Assign them.

Chapter 98. Agent groups

NXLog agent groups are used for agent management by grouping them. Under the hood agent tags are used for this purpose in little different manner. This chapter discusses the GUI configuration and management frontend provided by NXLog Manager.

To list the available NXLog agent groups, click on the Agent groups tab in [AGENTS](#) page. A list similar to the following should appear:

Group name	Description
linux-group	Linux group
windows-group	Windows group

Showing 1 to 2 of 2 entries

Add Help

This list contains the following information:

Group name

The group name is the unique name of a NXLog agent group. Clicking on the name will load the [agents](#) which are tagged by it.

Description

This is the NXLog agent group detailed description.

On the bottom of the list there is a row of buttons which can be used to manage the groups.

Add

[Add](#) a new group.

98.1. Agent list in a group

In agent list similar to the following should appear:

Group linux-group											
Show 100 entries											
Status	Agent name	Version	Host	Started	Load	Mem. usage	Received	Received today	Processing	Sent	Sent today
<input type="checkbox"/>	debian-m4aBt-m	3.0.1706	192.168.0.109	2016-10-07 13:37	1.5		8.48M		3		31
<input type="checkbox"/>	test	Unknown							0		3

Showing 1 to 2 of 2 entries

Refresh status Start Stop Update and reload Configure Download config View log Issue certificate Renew certificate Delete group Add agents Help

In agent list under a group page except the standard agent management buttons, there are additional buttons which can be used to manage this group.

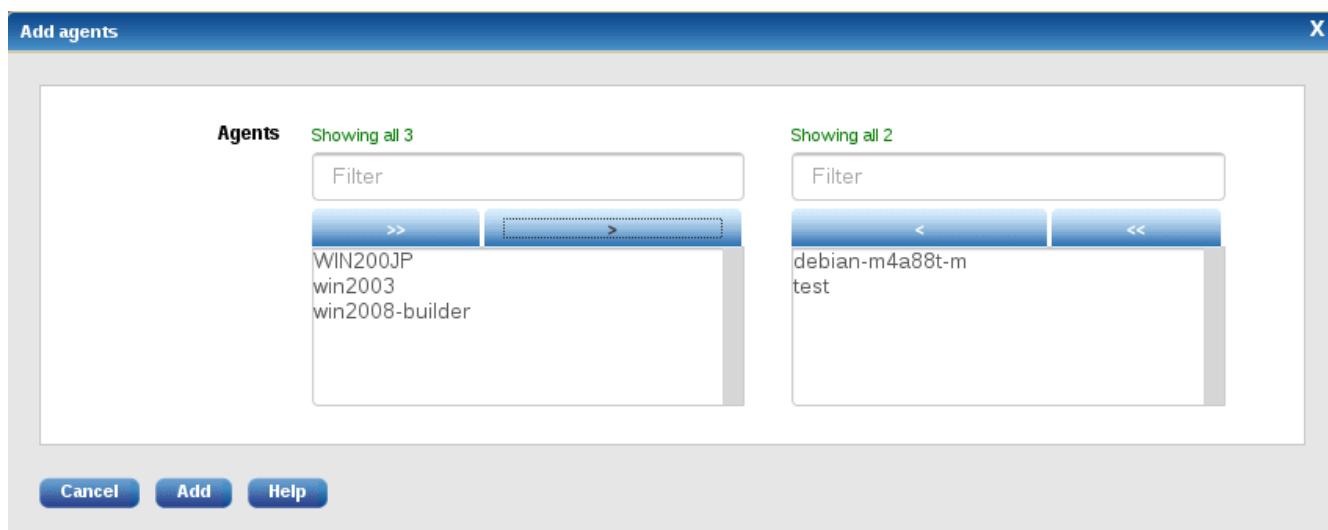
Delete group

Delete this group/tag.

Add agents

Add agents to this group.

To add agents to this group, click on the [Add agents](#) button. An Add agents dialog will appear:



Select the desired agents and click Add. The selected agents will be added to the list in this group.

Chapter 99. Certificates

NXLog Manager uses X509 certificates for various security purposes and has a built-in PKI system to manage these.

99.1. Listing certificates

To list the available certificates, click on the **CERTIFICATES** menu item under the ADMIN menu. A list similar to the following will appear.

Certificates list

<input type="checkbox"/>	Name	Type	Activation	Expiration	Status	Private Key	Search: <input type="text"/>
<input type="checkbox"/>	▶ nxm-ca	CA	2014-10-16 15:54	2015-10-16 00:00	VALID	YES	
<input type="checkbox"/>	▶ 192.168.56.102	CERT	2015-05-19 17:38	2016-05-19 00:00	VALID	YES	
<input type="checkbox"/>	▶ AgentManager	CERT	2014-10-16 15:56	2015-10-16 00:00	VALID	YES	
<input type="checkbox"/>	▶ test	CERT	2015-05-19 17:39	2016-05-19 17:39	VALID	YES	
<input type="checkbox"/>	▶ WIN2000JP	CERT	2015-05-19 17:40	2016-05-19 17:40	VALID	YES	
<input type="checkbox"/>	▶ win2003	CERT	2015-05-19 17:40	2016-05-19 17:40	VALID	YES	
<input type="checkbox"/>	▶ win2008-builder	CERT	2015-06-04 14:17	2016-06-04 00:00	VALID	YES	
<input type="checkbox"/>	▶ win2008-builder	CERT	2015-05-19 17:40	2016-05-19 17:40	REVOKED	YES	

Showing 1 to 8 of 8 entries

[Add new CA](#) [Create certificate](#) [Import](#) [Export](#) [Revoke](#) [Renew](#) [Delete](#) [Help](#)

The table contains the following information.

Name

The certificate subject name.

Type

This can be either CA or CERT.

Activation

The time and date after the certificate is valid.

Expiration

The time and date before the certificate is valid.

Status

The status of the certificate can be VALID, REVOKED and EXPIRED.

Private Key

This field indicates whether the private key pair of the certificate is available or not.

The certificate list shows entries in a hierarchical (tree) structure. Certificates (and sub-CAs) will be rooted under the CA which was used to sign it.

If the PKI system does not have any certificates, you will need to create a CA first.

99.2. Creating a CA

The certificate authority is used to issue and sign certificates and can be used to later verify the trust relationship. To be able to create certificates, a CA is required. To create a CA cert, click on the Add new CA on the certificate list page. The following screenshot shows the certificate creator form.

Create CA

*Name:

Add next

Country: country

State: state

City: city

Organization: organization

Org. Unit: organization unit

Valid To: 2016-06-06

Use as default CA Jun 2016

Create **Help**

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

Some values will be already filled in if the [Certificate settings](#) are configured. After clicking Create, the new CA should appear.

99.3. Creating a certificate

Create certificate

*Name:

Country Hungary

State Budapest

City Budapest

Organization

Org. Unit

Valid To

CA ROOTCA

Certificate purpose Agent Details (show/hide)

Create

Some values will be already filled in if the [Certificate settings](#) are configured. Fill in the name (certificate subject), expiry and select the certificate purpose. It is possible to customize the certificate purpose flags, but this is not required if the certificate will be used only within NXLog Manager and with NXLog. After clicking Create, the new certificate should appear. The following screenshot shows the information page of a certificate.

Certificate general information

Issuer	CN= [REDACTED]
Subject	CN= [REDACTED]
Serial Number	1333656428
Signature Hash Algorithm	SHA1withRSA
Purposes	SSL client SSL server
Uses	
Extended Key Usage	
Thumb Print	fa fd 59 50 8c 2a 2a 14 77 99 61 a2 b3 32 58 42 6d b7 b2 fd
Valid From	2012-04-05 22:07:08
Valid To	2020-04-01 00:00:00
Status	VALID
Private Key	YES
Extensions	<pre>NetscapeCertType [criticality: false] ssl_client: true ssl_server: true SubjectKeyIdentifier [criticality: false] KeyIdentifier: 6a 2c bb ad c8 ba ba 0e 82 24 f1 a9 4f 72 36 f5 83 1a 5a 4f AuthorityKeyIdentifier [criticality: false] KeyIdentifier: 50 3e 83 f3 cb 1b 0a be 39 2a d5 75 eb 82 3a 89 b3 c1 47 48 auth_name: null serial_number: null BasicConstraints [criticality: false] is_ca: false path_len: -1</pre>
Export	Revoke

99.4. Exporting

To export a certificate, click Export on the certificate general information page or below the certificate list after

selecting and entry. The following options will appear.

Export Certificate

Certificate Name win2003

Format

- Certificate in PEM format
- Private key in PEM format
- Certificate in DER format
- Public key in PEM format
- Certificate and private key in PKCS#12 format

Export **Back**

In order to support external certificate tools and PKI systems, certificates can be exported in different formats. The NXLog agents use PEM formatted X509 certificates.

99.5. Importing

Import Certificate

Please chose an import option:

- Certificate in PEM format
- Certificate and private key in PEM format
- Certificate and private key in PKCS#12 format
- Certificate Revocation List(CRL)

***Certificate:** **Browse...**

***Key file:** **Browse...**

Password:

Import

In order to support external certificate tools and PKI systems, certificates can be imported in different formats.

99.6. Revoking and deleting certificates

It is not possible to delete a certificate unless it is revoked. It is recommended not to delete certificates. If the PKI system does not contain the certificate of an NXLog agent and the presented certificate is authenticated, the connection will be accepted unless the certificate is found in the local PKI system and is revoked.

99.7. Renewing certificates

This operation will issue a new certificate which can be used to replace an existing one which has already expired, will shortly expire or is revoked.

NOTE

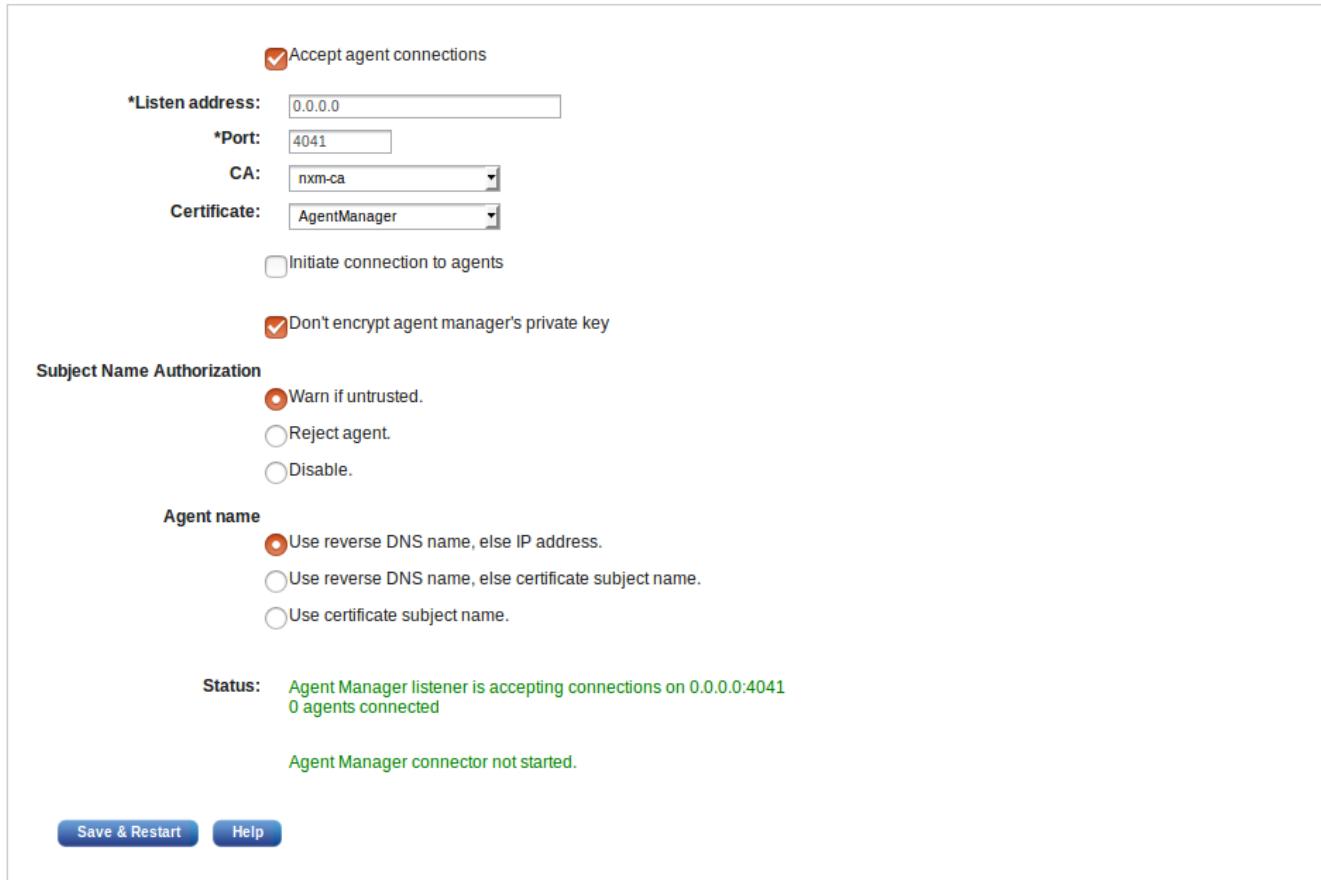
Generally it is not a good idea to have multiple valid certificates with the same subject. If a certificate has been superseeded by a new one (e.g. already pushed to the agent), make sure to revoke the former.

Chapter 100. Settings

To configure various system components, click on the **SETTINGS** menu item under the ADMIN menu. Each tab is discussed in the successive sections.

100.1. Agent Manager

The agent manager is responsible for connecting to the NXLog agents or accepting connections to establish a secure trusted channel which is used to manage and administer the agents remotely. Each NXLog agent is queried by the agent manager every 60 seconds for status information.



The above screenshot shows the Agent manager tab where its parameters can be configured.

The agent manager can both accept and initiate connections to the agents. Enable the Accept agent connections checkbox to let the agent manager accept incoming connections from agents.. Enable the Initiate connection to agents checkbox to let the agent manager initiate the connection.

NOTE

For these settings to work, the agents must be configured accordingly. See the [Agent Connection type](#) configuration parameter.

These options have the following configuration parameters:

Listen address

When Accept agent connections is requested, the IP address of the interface must be specified. Use 0.0.0.0 to listen on all interfaces.

Port

When Accept agent connections is requested, the port number must be specified where the agent manager will listen on for incoming connections.

CA

The CA configured here is used to verify the certificate presented by the NXLog agent during the SSL handshake.

Certificate

The certificate configured here will be used to authenticate to the NXLog agent during the SSL handshake.

For security reasons certificate private keys in the database are stored in an encrypted form. These are encrypted with a master key which is accessible to users who have ROLE_ADMINISTRATOR and/or ROLE_CERTIFICATE access rights. The agent manager's private key is required to be able to establish the trusted control connection with the agents. Enable the Don't encrypt agent manager's private key option for the system to be able to operate in an unattended mode. Otherwise the agent manager connection will only work after a reboot/restart after a successful admin login.

Another security option is *Subject Name Authorization*. *Subject Name Authorization* refers to the check that happens when the TLS connection is established with the agent and the manager looks at the CN in the certificate subject and checks whether this matches the reverse DNS. This is to prevent malicious agents connecting with a stolen certificate. The *Agent name* setting that follows is to select what data to use as the "agent name". These two options are useful on networks with DHCP assigned addresses where the agent may have different IP addresses. There are 3 options:

Warn if untrusted.

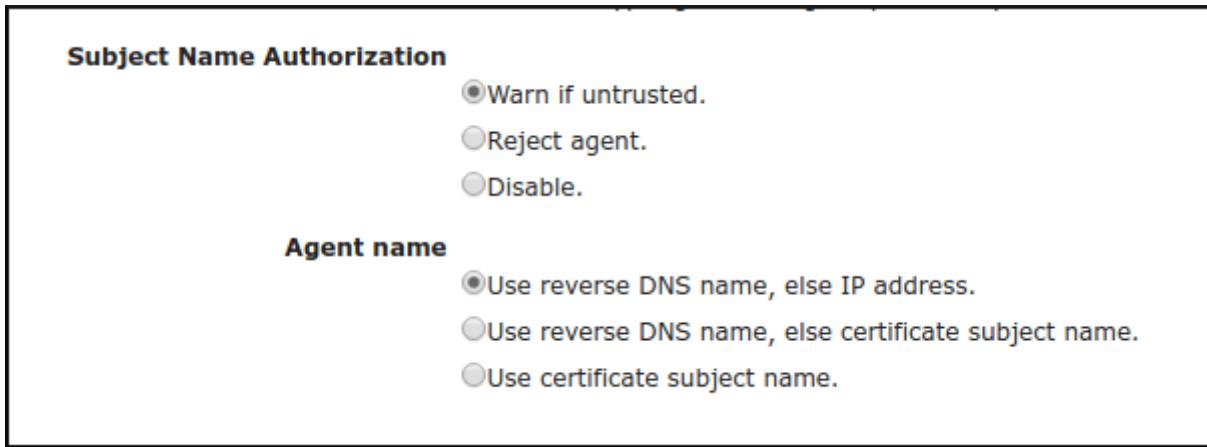
When this option is selected, agent manager will accept agents which try to authorize with Subject Name other than their reverse DNS, and will mark them as Forged.

Reject agent.

When this option is selected, agent manager will reject agents which try to authorize with Subject Name other than their reverse DNS.

Disable.

When this option is selected, agent manager will ignore the mismatch between Subject Name and reverse DNS for connected agents.



Due to Subject Name Authorization and the specifics of some networks, like NAT for example, agent manager must have some policy for names of connected agents which will appear on the [Agent list](#). Agent manager supports 3 options for Agent name:

Use reverse DNS name, else IP address.

When this option is selected, agent manager will try to resolve the Fully Qualified Domain Name of connected agents. If resolving fails, it'll use agent's IP address.

Use reverse DNS name, else certificate subject name.

When this option is selected, agent manager will try to resolve the Fully Qualified Domain Name of connected agents. If resolving fails, it'll use agent certificate's Subject Name.

Use certificate subject name.

When this option is selected, agent manager will always use agent certificate's Subject Name. This option is the only reasonable choice for NAT networks.

NOTE

When one of the last 2 options is selected and a NXLog agent doesn't authorize with valid client certificate, but the manager demands Subject Name, the agent will be rejected.

Click Save & Restart to apply the changes. The Status field will display the status of the agent manager.

100.2. Certificates

This form is divided in two sections: Certificate defaults and Certificates provider:

Agent manager		Certificates	Mail	Config backup	License	User Settings												
Certificate defaults <table border="1"> <tr><td>Country:</td><td>country</td></tr> <tr><td>State:</td><td>state</td></tr> <tr><td>City:</td><td>city</td></tr> <tr><td>Organization:</td><td>organization</td></tr> <tr><td>Org. unit:</td><td>organization unit</td></tr> <tr><td>Default CA:</td><td>- Select Certificate -</td></tr> </table>							Country:	country	State:	state	City:	city	Organization:	organization	Org. unit:	organization unit	Default CA:	- Select Certificate -
Country:	country																	
State:	state																	
City:	city																	
Organization:	organization																	
Org. unit:	organization unit																	
Default CA:	- Select Certificate -																	
Certificates provider <table border="1"> <tr><td>Provider type:</td><td>Database</td></tr> <tr><td>Add</td><td>Edit</td></tr> </table>							Provider type:	Database	Add	Edit								
Provider type:	Database																	
Add	Edit																	
<input type="button" value="Save"/> <input type="button" value="Help"/>																		

Certificate defaults

This form can be used to set common parameters which are used during certificate creation.

Certificate

The Certificates provider option makes it possible to use a PKCS11 compliant backend to store certificates and private keys instead of using the configuration database. The PKCS11 API is implemented by most smart cards and HSM devices which can be used to securely store private keys.

100.3. Mail

To be able to send notification emails, an SMTP server is required. The Mail tab provides a form where the SMTP server settings can be specified.

***Hostname:** mail.localhost.com

Protocol: smtp

***Port:** 25

Username: [empty]

Password: [empty]

***Default from address:** admin@localhost.com

Subject prefix: [empty]

Save **Help** **Send test email to:** [empty]

100.4. Config backup

The full NXLog Manager configuration can be backed up to an encrypted file. The configuration can be restored using a backup file on the same form. This configuration backup can be scheduled to make it run automatically at a specific time. The system will send an email notification if an email address is provided.

***Certificate:** -- Select Certificate --

Enable email notification: Disabled because you do not have a mail server properly configured.

To addresses: [empty]

Schedule:

Save **Help**

***Restore certificate for decryption:** -- Select Certificate --

Enable email notification: Disabled because you do not have a mail server properly configured.

To addresses: [empty]

***Backup file:** No file selected.

Save **Help**

100.5. License

The License tab provides a form where the license file can be uploaded and the license details are shown.

Agent manager Certificates Mail Config backup **License** User Settings

License conditions

Expiry 2011-12-31
IssuedBy nxsec.com
IssuedFor log4ensics developers

Upload license file

License file

No file selected.

If the license is invalid or expired, a message will be displayed in a horizontal red band as shown in the following image.



100.6. User Settings

This form is divided in two sections: *Settings* and *Change password*:

Agent manager Certificates Mail Config backup License **User Settings**

Settings

*Full name	Administrator
E-mail address	
Language	en
UI Theme	blue

Change password

Old password	
New password	
Confirm new password	

Save **Help**

Settings

The User Settings tab allows to the logged in user to change his/her name, email address, user interface language and theme. The email address will be used for system notifications.

Change password

The Change Password tab allows to the logged in user to change his/her password.

Chapter 101. Users, roles and access control

NXLog Manager has a sophisticated user and role management system which makes it possible to allow or deny access to certain features and/or resources such as reports or the log data itself.

101.1. Users

The user management interface can be accessed by clicking on the **USERS** menu item under the ADMIN menu. The default installation has only the admin user. To add a new user, click on the Add User below the left panel as seen in the following screenshot.

Manage Users

The screenshot shows the 'Manage Users' dialog window. On the left, a sidebar titled 'Users' lists two users: 'admin' and 'John'. The main area is divided into two tabs: 'Details' and 'Assigned Roles'. The 'Details' tab displays the user information for 'admin': 'Administrator' and 'This user is enabled.' The 'Assigned Roles' tab shows the role 'ROLE_ADMINISTRATOR'. At the bottom of the window are buttons for 'Add User', 'Help', 'Edit', and 'Delete User'.

The following dialog window will appear where the user's details and credentials can be provided.

Add User

*User ID

Full Name

Email Address

*Password

*Confirm Password

Enable this user

Cancel **Add** **Help**

Make sure to toggle the Enable this user checkbox. After clicking on Submit, the newly created user should appear in the list. Select the user in the list and click Edit. This lets you change the user information and user's roles.

The assigned roles are shown in the second block in the right hand side. Click Edit to modify the user's roles. The following fragment will appear:

Manage Users

Users		Details
admin John		User Info User ID: John Password: <input type="password"/> Full Name: John Confirm Password: <input type="password"/> Email Address: john@nxlog.org <input checked="" type="checkbox"/> Enable this user
Assigned Roles Showing all 10 Filter: <input type="text"/> >> > < << ROLE_ADMINISTRATOR RW ROLE_REPORT_ADMIN RW ROLE_REPORT RW ROLE_ANALYSIS RW ROLE_EVENTDB_ADMIN RW ROLE_DBSEARCH RW		
Showing all 2 Filter: <input type="text"/> < << ROLE_PATTERN RO ROLE_AGENT RW		
Add User Help Save Cancel Help		

Select the roles needed for this user. When all required roles are added, click Save.

NOTE

By default the roles are with read-write permissions. To restrict certain roles to read-only, click once at the selected role.

101.2. Roles

The role management interface can be accessed by clicking on the **ROLES** menu item under the ADMIN menu.

By default NXLog Manager comes with a built-in set of roles which are listed in the following screenshot on the left.

Manage Roles

The screenshot shows the 'Manage Roles' interface. On the left, a sidebar titled 'Roles' lists ten built-in roles: ROLE_ADMINISTRATOR, ROLE_REPORT_ADMIN, ROLE_REPORT, ROLE_ANALYSIS, ROLE_EVENTDB_ADMIN, ROLE_DBSEARCH, ROLE_FIELD, ROLE_PATTERN, ROLE_CORRELATION, ROLE_AGENT, ROLE_CERTIFICATE, and ROLE_READONLY. At the bottom of this sidebar are 'Add Role' and 'Help' buttons. On the right, a larger panel titled 'Details' shows the details for the selected role, 'ROLE_ADMINISTRATOR'. It includes a sub-section titled '1 Users with this role' containing the user 'admin'. At the bottom of this panel are 'Delete Role' and 'Edit' buttons.

These built-in roles are as follows:

ROLE_ADMINISTRATOR

All functions are available to the user who has this role.

ROLE_AGENT

The **AGENTS** menu is not visible and the user may not configure, view or manage the agents without this role (or ROLE_ADMINISTRATOR).

ROLE_CERTIFICATE

The user may not access the PKI system (**CERTIFICATES** menu) and issue, modify or access certificates in any way without this role (or ROLE_ADMINISTRATOR).

ROLE_CORRELATION

The user may not create, modify or access any correlation rules and the **CORRELATION** menu without this role (or ROLE_ADMINISTRATOR).

ROLE_PATTERN

The user may not create, modify or delete **patterns** without this role (or ROLE_ADMINISTRATOR).

ROLE_READONLY

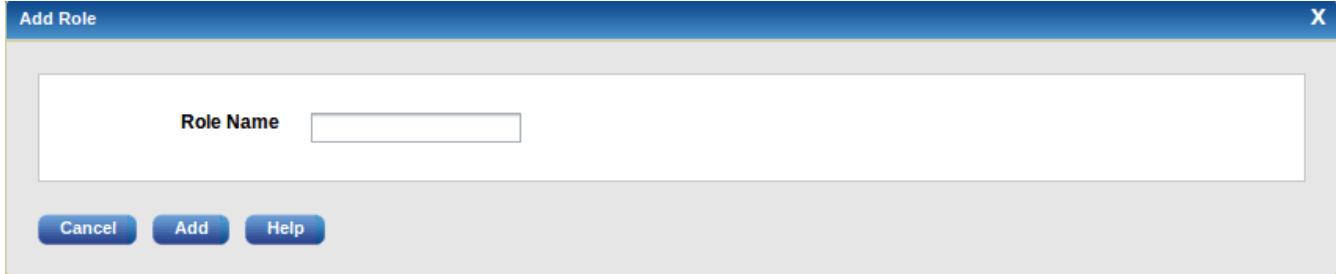
This is a special role which denies any modification to the system by the user who has this.

ROLE_USER_MANAGEMENT

The user may not access the user and role management system to create, modify or delete users and roles without this role (or ROLE_ADMINISTRATOR).

The above built-in roles may not be removed from the system.

It may be necessary to create special roles for more sophisticated access control Click Add role below the role list. The following dialog window will appear.

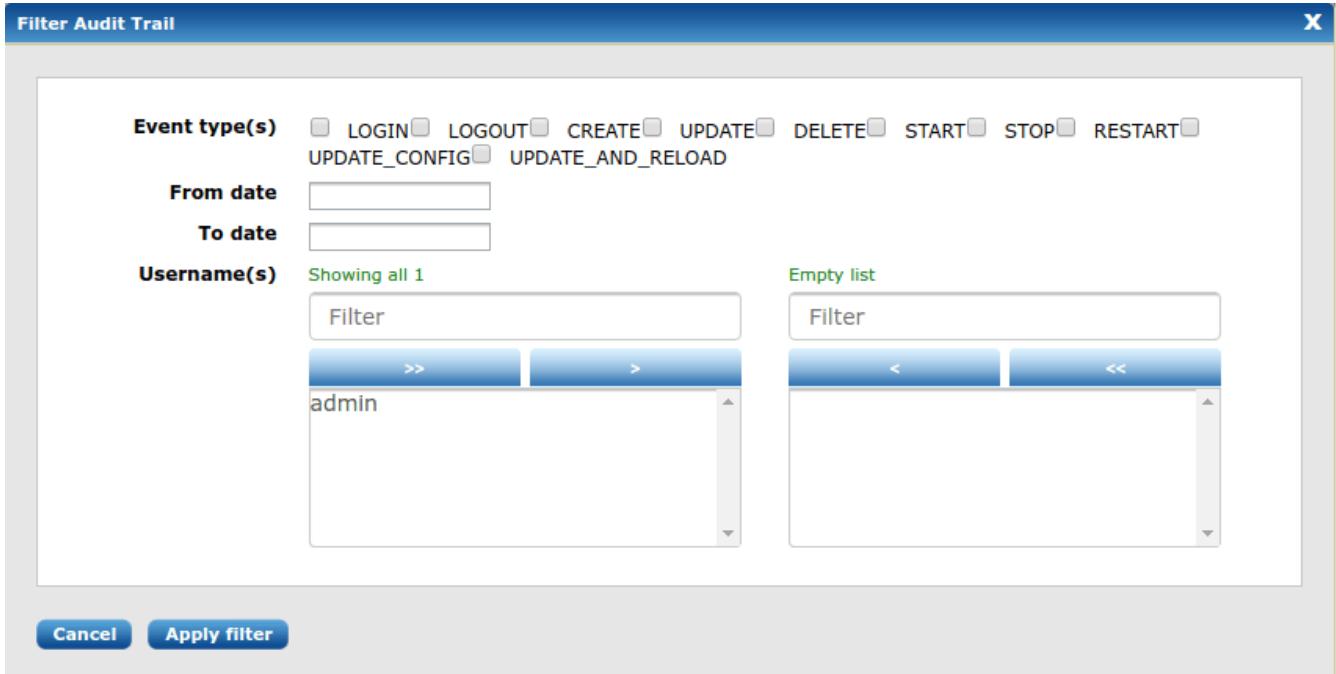


Click Submit after filling in the role's name. It should appear in the list.

101.3. Audit Trail

NXLog Manager keeps a record of all the events that happened so far, also known as an audit trail. The audit trail can be accessed by clicking on the AUDIT TRAIL menu item under the ADMIN menu. The user is presented with a list of events sorted in chronological order. The fields include the event date, event type, username, manager address, user address and details about the event. By clicking on any of the column headers the events can be resorted by ascending or descending order. Clicking on the plus symbol in the Details column the Details field will collapse showing more information.

On the top of the list there is a [**Filter audit trail**] button which can be used for filtering the events on the list. When clicked, the following dialog appears:



Here audit events can be filtered by 3 criteria: the event type, a time period and finally the username. Clicking [**Apply filter**] will refresh the event list with only events matching the filtering criteria. For example selecting DELETE Event types will show the following:

Filter audit trail						Clear filter
Filtered by: Event types([DELETE]));						Search: <input type="text"/>
Event date	Event type	Username	Manager address	User address	Details	
2017-10-13 11:17	DELETE AGENT INSTANCE	admin	172.17.0.1	0:0:0:0:0:0:1	["name": "localhost"]	
2017-12-07 12:39	DELETE AGENT INSTANCE	admin	172.17.0.1	0:0:0:0:0:0:1	["name": "10.15.173.2"]	
2017-12-07 20:50	DELETE AGENT INSTANCE	admin	192.168.122.1	0:0:0:0:0:0:1	["name": "192.168.122.1"]	
2017-12-07 20:51	DELETE AGENT INSTANCE	admin	192.168.122.1	0:0:0:0:0:0:1	["name": "localhost"]	

When a filter is applied, the [Clear filter] button appears which can be used to discard the applied filter to show all audit events.

Chapter 102. RESTful Web Services

NXLog manager provides a Representational state transfer (REST) interface or RESTful web service API that makes it possible to access agent information or configure the system without using the UI.

The base URL to access the REST API is <http://hostname:port/nxlog-manager/mvc/restservice> and depending on the service either GET, POST, PUT or DELETE requests should be used. The API responses are in XML.

NOTE Throughout this chapter the base URL will be substituted with [B_URL] for convenience.

The following services are available:

- [agentmanager](#) - Verify that NXLog Manager is up and running (GET).
- [appinfo](#) - Provides information about the NXLog Manager (GET).
- [agentinfo](#) - Provides information about the NXLog Agents (GET).
- [addagent](#) - Add a new Agent (POST).
- [modifyagent](#) - Modify an existing Agent (PUT).
- [deleteagent](#) - Delete an existing Agent (DELETE).
- [certificateinfo](#) - Retrieve certificate information (GET).
- [createfield](#) - Create NXLog fields (POST).

102.1. Agentmanager

This service is useful to verify the NXLog Manager is up and running. This is a **GET** request with URL [\[B_URL\]/agentmanager](#) and no additional parameters. This service can also be used if the "Don't encrypt agent manager's private key" is not enabled on the [settings](#) tab and the NXLog Manager service has been restarted (or after a reboot). A REST call of a user who has ROLE_ADMINISTRATOR and/or ROLE_CERTIFICATE access rights can decrypt the master key thus, enabling the agent manager to establish the trusted control connection with the agents.

102.2. Appinfo

This service provides information about a running NXLog Manager. This is a **GET** request with URL [\[B_URL\]/appinfo](#) and no additional parameters.

Sample Output

```
?xml version='1.0' encoding='UTF-8'?
<result servicename="appinfo">
<values>
<applicationinfo>
<14eUptime>17143265</14eUptime>
<licenseState>LICENSED/Expired</licenseState>
<licenseExpireDate>2011-12-30 22:00:00.0 UTC</licenseExpireDate>
<appVersion>5.0</appVersion>
<appRevisionNumber>4895</appRevisionNumber>
</applicationinfo>
</values>
```

The information that can be collected from this service is the NXLog Manager's uptime, license state and expiration date, version and revision.

102.3. Agentinfo

This service provides information about NXLog Agents registered with the NXLog Manager. This is a **GET** request with URL **[B_URL]/agentinfo** that can take additional parameters. The response can be filtered by the name or the state of the agent with the options 'agentname' and 'agentstate'. Those two parameters can not be combined together, unlike the third parameter 'agentwithmodules' that will also include module information with the agent information. For example to get information for both the agents and the modules for all agents with state 'ONLINE' the following REST call can be used **[B_URL]/agentinfo?agentstate=ONLINE&agentwithmodules=true**. Refer to the [agents](#) chapter for more information.

Sample Output

```
?xml version='1.0' encoding='UTF-8'?>
<result servicename="agentinfo">
  <values>
    <agent>
      <name>192.168.122.1</name>
      <version>3.99.2866</version>
      <status>ONLINE</status>
      <load>0.16</load>
      <host>192.168.122.1</host>
      <started>2017-12-15 16:36:23.974 UTC</started>
      <memUsage>7442432.0</memUsage>
      <received>2</received>
      <processing>0</processing>
      <sent>2</sent>
      <sysinfo>OS: Linux, Hostname: voyager, Release: 4.4.0-103-generic, Version: #126-Ubuntu SMP Mon Dec 4 16:23:28 UTC 2017, Arch: x86_64, 4 CPU(s), 15.7Gb memory</sysinfo>
      <modules>
        <module>
          <name>in_int</name>
          <module>im_internal</module>
          <type>INPUT</type>
          <isRunning>true</isRunning>
          <received>2</received> <processing>0</processing>
          <sent>2</sent>
          <dropped>0</dropped>
          <status>RUNNING</status>
        </module>
        <module>
          <name>null_out</name>
          <module>om_null</module>
          <type>OUTPUT</type>
          <isRunning>true</isRunning>
          <received>2</received>
          <processing>0</processing>
          <sent>2</sent>
          <dropped>0</dropped>
          <status>RUNNING</status>
        </module>
      </modules>
    </agent>
  </values>
```

102.4. Addagent

This service adds a new NXLog Agent to the list of existing Agents. This is a **POST** request with URL **[B_URL]/addagent** that can take several additional parameters. The only mandatory parameter is 'agentname'

that is the name for the new agent. The optional parameter 'connectionmode' can be used to change the connection type of the Agent, from the default that is 'CONNECT_TO', to either 'UNMANAGED' or 'LISTEN_FROM'. The 'connectionport' parameter can be used to change the default port of the manager that is 4041, this parameter can only be used for managed connection types. The 'connectionaddress' parameter can be used to set the IP address the manager will either 'CONNECT_TO' or 'LISTEN_FROM', the default value is localhost. The 'loglevel' parameter can be used to set the log level, values can be DEBUG, INFO,WARNING, ERROR, CRITICAL. The 'logofile' parameter is used to enable the agent to use the local 'nxlog.log' file.

To create agent clones, the 'agentname' parameter can be specified more than once with unique agent names.

Refer to the [agents](#) chapter for more information.

To create an Agent template instead of an Agent the 'agenttemplate=true' parameter can be used. If multiple agent names are specified when creating a template the first one will be the name of the template and the rest will be agents based on this template.

Refer to the [templates](#) chapter for more information.

Creating a new Agent for example with the following REST call:
[\[B_URL\]/addagent?agentname=Justatest&connectionmode=LISTEN_FROM](#), will return the following XML message that includes the Agent configuration in base64 encoded format.

Sample Output

```
?xml version='1.0' encoding='UTF-8'?
<result servicename="addagent">
  <values>
    <addagent>
      <configuration># PD94bWwgdmVyc2lvbj0iMS4wIiB1bmNvZGluZz0iVVRLTgiPz4KPGFnZW50PgogICAgPG5hbWU+
# SnVzdGF0ZXN0PC9uYW1lPgogICAgPG5zMTPnbG9iYWWtY29uZmlnIHhtbG5z0m5zM0iaHR0cDov
# L2NhC3RvcI5leG9sYWIub3JnLyI+CiAgICAgICAgPGNlcnQtaWQ+MTg8L2NlcnQtaWQ+CiAgICAg
# ICAGPGxvZy1sZXZlCB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1h
# LWluc3RhbmNlIgogICAgICAgICAgICB4bWxuczpqYXZhPSJodHRwOi8vamF2YS5zdW4uY29tIiB4
# c2k6dHlwZT0iamF2YTpqYXZhLmxhbmcuU3RyaW5nIj5JTkZPPC9sb2ctbGV2ZWw+CiAgICAgICAg
# PGlzLWxvZy10by1maWx1PnRydWU8L21zLWxvZy10by1maWx1PgogICAgICAgIDxb25uZWN0aW9u
# LW1vZGUKICAgICAgICAgeG1sbnM6eHNpPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1hNTFNj
# aGVtYS1pbnN0YW5jZSIKICAgICAgICAgICAgeG1sbnM6amF2YT0iaHR0cDovL2phdmEuc3VuLmNv
# bSIgeHNpOnR5cGU9ImphdmE6amF2YS5sYW5nL1N0cmcluZyI+TE1TVEV0X0ZST008L2Nvbm51Y3Rp
# b24tbW9kZT4KICAgICAgICA8Y29ubmVjdGlvbi1wb3J0PjA8L2Nvbm51Y3Rpb24tcG9ydD4KICAg
# IDwvbnMx0mdsb2JhbC1jb25maWc+CjwvYWdlbnQ+Cg==

    </configuration>
    </addagent>
  </values>
```

102.5. Modifyagent

This service modifies the configuration of an existing Agent. This is a [PUT](#) request with URL [\[B_URL\]/modifyagent](#). This service has the same parameters as the addagent service, except from the 'agenttemplate' parameter.

102.6. Deleteagent

This service deletes an existing Agent. This is a [DELETE](#) request with URL [\[B_URL\]/deleteagent](#). The only parameter required for this service is the 'agentname' parameter.

102.7. Certificateinfo

This safe service can retrieve certificate information from the NXLog Manager. This is a **GET** request with URL **[B_URL]/certificateinfo**. Without any parameters the service will list all certificate information however, parameter 'expirein' can be used to list only certificates that will expire in the given number of days.

As an example the following call will list certificates expiring in one month. **[B_URL]/certificateinfo?expirein=30**, if no certificates expire at that time an empty result is returned.

Sample Output

```
?xml version='1.0' encoding='UTF-8'?
<result servicename="certificateinfo">
  <values>
    <ok>
      <message>The result is empty!</message>
    </ok>
  </values>
```

102.8. Createfield

This service will create "fields" in NXLog Manager. This is a POST request with URL **[B_URL]/createfield** and there are several parameters. Parameter 'name' is the name of the field and must be a unique identifier. Parameter 'type' is the field type and must be one of the types: STRING, INTEGER, BINARY, IP4ADDR, IP6ADDR, BOOLEAN, DATETIME. Parameter 'description' is a short description of the field. Parameters 'persist' and 'lookup' can be 'true' or 'false'. Refer to the [fields](#) chapter for more information.

The following REST call will create a 'TEST' field of type 'STRING' both persistent and lookup enabled.

[B_URL]/createfield?name=TEST&type=STRING&description=Just a string&persistent=true&lookup=true

Sample Output

```
?xml version='1.0' encoding='UTF-8'?
<result servicename="createfield">
  <values>
    <ok>
      <message>OK</message>
    </ok>
  </values>
```