# Appenders

Appenders are responsible for delivering LogEvents to their destination. Every Appender must implement the Appender (../log4j-core/apidocs/org/apache/logging/log4j/core/Appender.html) interface. Most Appenders will extend AbstractAppender (../log4j-core/apidocs/org/apache/logging/log4j/core/appender/AbstractAppender.html) which adds Lifecycle (../log4j-core/apidocs/org/apache/logging/log4j/core/LifeCycle.html) and Filterable (../log4j-core/apidocs/org/apache/logging/log4j/core/filter/Filterable.html) support. Lifecycle allows components to finish initialization after configuration has completed and to perform cleanup during shutdown. Filterable allows the component to have Filters attached to it which are evaluated during event processing.

Appenders usually are only responsible for writing the event data to the target destination. In most cases they delegate responsibility for formatting the event to a layout (layouts.html). Some appenders wrap other appenders so that they can modify the LogEvent, handle a failure in an Appender, route the event to a subordinate Appender based on advanced Filter criteria or provide similar functionality that does not directly format the event for viewing.

Appenders always have a name so that they can be referenced from Loggers.

In the tables below, the "Type" column corresponds to the Java type expected. For non-JDK classes, these should usually be in Log4j Core (../log4j-core/apidocs/index.html) unless otherwise noted.

## AsyncAppender

The AsyncAppender accepts references to other Appenders and causes LogEvents to be written to them on a separate Thread. Note that exceptions while writing to those Appenders will be hidden from the application. The AsyncAppender should be configured after the appenders it references to allow it to shut down properly.

By default, AsyncAppender uses java.util.concurrent.ArrayBlockingQueue (http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ArrayBlockingQueue.html) which does not require any external libraries. Note that multi-threaded applications should exercise care when using this appender as such: the blocking queue is susceptible to lock contention and our tests showed (../performance.html#asyncLogging) performance may become worse when more threads are logging concurrently. Consider using lock-free Async Loggers (async.html) for optimal performance.

AsyncAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| AppenderRef | String | The name of the Appenders to invoke asynchronously. Multiple AppenderRef elements can be configured. |
| blocking | boolean | If true, the appender will wait until there are free slots in the queue. If false, the event will be written to the error appender if the queue is full. The default is true. |
| shutdownTimeout | integer | How many milliseconds the Appender should wait to flush outstanding log events in the queue on shutdown. The default is zero which means to wait forever. |
| bufferSize | integer | Specifies the maximum number of events that can be queued. The default is 1024. Note that when using a disruptor-style BlockingQueue, this buffer size must be a power of 2.<br>When the application is logging faster than the underlying appender can keep up with for a long enough time to fill up the queue, the behaviour is determined by the AsyncQueueFullPolicy (../log4j-core/apidocs/org/apache/logging/log4j/core/async/AsyncQueueFullPolicy.html). |
| errorRef | String | The name of the Appender to invoke if none of the appenders can be called, either due to errors in the appenders or because the queue is full. If not specified then errors will be ignored. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| name | String | The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| includeLocation | boolean | Extracting location is an expensive operation (it can make logging 5 - 20 times slower). To improve performance, location is not included by default when adding a log event to the queue. You can change this by setting includeLocation="true". |
| BlockingQueueFactory | BlockingQueueFactory | This element overrides what type of BlockingQueue to use. See below documentation for more details. |

There are also a few system properties that can be used to maintain application throughput even when the underlying appender cannot keep up with the logging rate and the queue is filling up. See the details for system properties log4j2.AsyncQueueFullPolicy and log4j2.DiscardThreshold (configuration.html#log4j2.AsyncQueueFullPolicy).

A typical AsyncAppender configuration might look like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <File name="MyFile" fileName="logs/app.log">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
    </File>
    <Async name="Async">
      <AppenderRef ref="MyFile"/>
    </Async>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Async"/>
    </Root>
  </Loggers>
</Configuration>
```

Starting in Log4j 2.7, a custom implementation of BlockingQueue or TransferQueue can be specified using a BlockingQueueFactory (../log4j-core/apidocs/org/apache/logging/log4j/core/async/BlockingQueueFactory.html) plugin. To override the default BlockingQueueFactory, specify the plugin inside an <Async/> element like so:

```xml
<Configuration name="LinkedTransferQueueExample">
  <Appenders>
    <List name="List"/>
    <Async name="Async" bufferSize="262144">
      <AppenderRef ref="List"/>
      <LinkedTransferQueue/>
    </Async>
  </Appenders>
  <Loggers>
    <Root>
      <AppenderRef ref="Async"/>
    </Root>
  </Loggers>
</Configuration>
```

Log4j ships with the following implementations:

BlockingQueueFactory Implementations

| Plugin Name | Description |
|---|---|
| ArrayBlockingQueue | This is the default implementation that uses ArrayBlockingQueue (https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ArrayBlockingQueue.html). |
| DisruptorBlockingQueue | This uses the Conversant Disruptor (https://github.com/conversant/disruptor) implementation of BlockingQueue. This plugin takes a single optional attribute, spinPolicy, which corresponds to |
| JCToolsBlockingQueue | This uses JCTools (https://jctools.github.io/JCTools/) , specifically the MPSC (multiple producer single consumer) bounded lock-free queue. |
| LinkedTransferQueue | This uses the new in Java 7 implementation LinkedTransferQueue (https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/LinkedTransferQueue.html). Note that this queue does not use the bufferSize configuration attribute from AsyncAppender as LinkedTransferQueue does not support a maximum capacity. |

# CassandraAppender

The CassandraAppender writes its output to an Apache Cassandra (https://cassandra.apache.org/) database. A keyspace and table must be configured ahead of time, and the columns of that table are mapped in a configuration file. Each column can specify either a StringLayout (layouts.html) (e.g., a PatternLayout (layouts.html#PatternLayout)) along with an optional conversion type, or only a conversion type for org.apache.logging.log4j.spi.ThreadContextMap or org.apache.logging.log4j.spi.ThreadContextStack to store the MDC or NDC (thread-context.html) in a map or list column respectively. A conversion type compatible with java.util.Date will use the log event timestamp converted to that type (e.g., use java.util.Date to fill a timestamp column type in Cassandra).

CassandraAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| batched | boolean | Whether or not to use batch statements false. |
| batchType | BatchStatement.Type (http://docs.datastax.com/en/drivers/java/3.0/com/datastax/driver/core/BatchStatement.Type.html) | The batch type to use when using batch |
| bufferSize | int | The number of log messages to buffer |
| clusterName | String | The name of the Cassandra cluster to c |

| | | |
|---|---|---|
| columns | ColumnMapping[] | A list of column mapping configurations can have a conversion type specified b type is String. If the configured type is a api/apidocs/org/apache/logging/log4j/ut api/apidocs/org/apache/logging/log4j/sp (../log4j-api/apidocs/org/apache/logging be populated with the MDC or NDC res with java.util.Date, then the log timesta attribute is given, then its value will be u Otherwise, the layout or pattern specifie that column. |
| contactPoints | SocketAddress[] | A list of hosts and ports of Cassandra n addresses. By default, if a port is not sp Cassandra port of 9042 will be used. B |
| filter | Filter | A Filter to determine if the event should be used by using a CompositeFilter. |
| ignoreExceptions | boolean | The default is true, causing exceptions logged and then ignored. When set to f You must set this to false when wrappir |
| keyspace | String | The name of the keyspace containing t |
| name | String | The name of the Appender. |
| password | String | The password to use (along with the us |
| table | String | The name of the table to write log mess |
| useClockForTimestampGenerator | boolean | Whether or not to use the configured or TimestampGenerator (http://docs.datastax.com/en/drivers/jav By default, this is false. |
| username | String | The username to use to connect to Cas |
| useTls | boolean | Whether or not to use TLS/SSL to conn |

Here is an example CassandraAppender configuration:

```
<Configuration name="CassandraAppenderTest">
  <Appenders>
    <Cassandra name="Cassandra" clusterName="Test Cluster" keyspace="test" table="logs" bufferSize="10" batched="true">
      <SocketAddress host="localhost" port="9042"/>
      <ColumnMapping name="id" pattern="%uuid{TIME}" type="java.util.UUID"/>
      <ColumnMapping name="timeid" literal="now()"/>
      <ColumnMapping name="message" pattern="%message"/>
      <ColumnMapping name="level" pattern="%level"/>
      <ColumnMapping name="marker" pattern="%marker"/>
      <ColumnMapping name="logger" pattern="%logger"/>
      <ColumnMapping name="timestamp" type="java.util.Date"/>
      <ColumnMapping name="mdc" type="org.apache.logging.log4j.spi.ThreadContextMap"/>
      <ColumnMapping name="ndc" type="org.apache.logging.log4j.spi.ThreadContextStack"/>
    </Cassandra>
  </Appenders>
  <Loggers>
    <Logger name="org.apache.logging.log4j.cassandra" level="DEBUG">
      <AppenderRef ref="Cassandra"/>
    </Logger>
    <Root level="ERROR"/>
  </Loggers>
</Configuration>
```

This example configuration uses the following table schema:

```
CREATE TABLE logs (
    id timeuuid PRIMARY KEY,
    timeid timeuuid,
    message text,
    level text,
    marker text,
    logger text,
    timestamp timestamp,
    mdc map<text,text>,
    ndc list<text>
);
```

## ConsoleAppender

As one might expect, the ConsoleAppender writes its output to either System.out or System.err with System.out being the default target. A Layout must be provided to format the LogEvent.

ConsoleAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| follow | boolean | Identifies whether the appender honors reassignments of System.out or System.err via System.setOut or System.setErr made after configuration. Note that the follow attribute cannot be used with Jansi on Windows. Cannot be used with direct. |
| direct | boolean | Write directly to java.io.FileDescriptor and bypass java.lang.System.out/.err. Can give up to 10x performance boost when the output is redirected to file or other process. Cannot be used with Jansi on Windows. Cannot be used with follow. Output will not respect java.lang.System.setOut()/.setErr() and may get intertwined with other output to java.lang.System.out/.err in a multi-threaded application. *New since 2.6.2. Be aware that this is a new addition, and it has only been tested with Oracle JVM on Linux and Windows so far.* |
| name | String | The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| target | String | Either "SYSTEM_OUT" or "SYSTEM_ERR". The default is "SYSTEM_OUT". |

A typical Console configuration might look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Console name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="%m%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="STDOUT"/>
    </Root>
  </Loggers>
</Configuration>
```

# FailoverAppender

The FailoverAppender wraps a set of appenders. If the primary Appender fails the secondary appenders will be tried in order until one succeeds or there are no more secondaries to try.

FailoverAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| primary | String | The name of the primary Appender to use. |
| failovers | String[] | The names of the secondary Appenders to use. |
| name | String | The name of the Appender. |
| retryIntervalSeconds | integer | The number of seconds that should pass before retrying the primary Appender. The default is 60. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. |
| target | String | Either "SYSTEM_OUT" or "SYSTEM_ERR". The default is "SYSTEM_ERR". |

A Failover configuration might look like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingFile name="RollingFile" fileName="logs/app.log" filePattern="logs/app-%d{MM-dd-yyyy}.log.gz"
                 ignoreExceptions="false">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <TimeBasedTriggeringPolicy />
    </RollingFile>
    <Console name="STDOUT" target="SYSTEM_OUT" ignoreExceptions="false">
      <PatternLayout pattern="%m%n"/>
    </Console>
    <Failover name="Failover" primary="RollingFile">
      <Failovers>
        <AppenderRef ref="Console"/>
      </Failovers>
    </Failover>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Failover"/>
    </Root>
  </Loggers>
</Configuration>
```

# FileAppender

The FileAppender is an OutputStreamAppender that writes to the File named in the fileName parameter. The FileAppender uses a FileManager (which extends OutputStreamManager) to actually perform the file I/O. While FileAppenders from different Configurations cannot be shared, the FileManagers can be if the Manager is accessible. For example, two web applications in a servlet container can have their own configuration and safely write to the same file if Log4j is in a ClassLoader that is common to both of them.

FileAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| append | boolean | When true - the default, records will be appended to the end of the file. When set to false, the file will be cleared before new records are written. |
| bufferedIO | boolean | When true - the default, records will be written to a buffer and the data will be written to disk when the buffer is full or, if immediateFlush is set, when the record is written. File locking cannot be used with bufferedIO. Performance tests have shown that using buffered I/O significantly improves performance, even if immediateFlush is enabled. |
| bufferSize | int | When bufferedIO is true, this is the buffer size, the default is 8192 bytes. |
| createOnDemand | boolean | The appender creates the file on-demand. The appender only creates the file when a log event passes all filters and is routed to this appender. Defaults to false. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| fileName | String | The name of the file to write to. If the file, or any of its parent directories, do not exist, they will be created. |
| immediateFlush | boolean | When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance.<br>Flushing after every write is only useful when using this appender with synchronous loggers. Asynchronous loggers and appenders will automatically flush at the end of a batch of events, even if immediateFlush is set to false. This also guarantees the data is written to disk but is more efficient. |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| locking | boolean | When set to true, I/O operations will occur only while the file lock is held allowing FileAppenders in multiple JVMs and potentially multiple hosts to write to the same file simultaneously. This will significantly impact performance so should be used carefully. Furthermore, on many systems the file lock is "advisory" meaning that other applications can perform operations on the file without acquiring a lock. The default value is false. |
| name | String | The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| filePermissions | String | File attribute permissions in POSIX format to apply whenever the file is created.<br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view.<br>Examples: rw------- or rw-rw-rw- etc... |

| | | |
|---|---|---|
| fileOwner | String | File owner to define whenever the file is created.<br><br>Changing file's owner may be restricted for security reason and Operation not permitted IOException thrown. Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file if _POSIX_CHOWN_RESTRICTED (http://www.gnu.org/software/libc/manual/html_node/Options-for-Files.html) is in effect for path.<br><br>Underlying files system shall support file owner (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/FileOwnerAttributeView.html) attribute view. |
| fileGroup | String | File group to define whenever the file is created.<br><br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view. |

Here is a sample File configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <File name="MyFile" fileName="logs/app.log">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
    </File>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="MyFile"/>
    </Root>
  </Loggers>
</Configuration>
```

# FlumeAppender

*This is an optional component supplied in a separate jar.*

Apache Flume (http://flume.apache.org/index.html) is a distributed, reliable, and available system for efficiently collecting, aggregating, and moving large amounts of log data from many different sources to a centralized data store. The FlumeAppender takes LogEvents and sends them to a Flume agent as serialized Avro events for consumption.

The Flume Appender supports three modes of operation.

1. It can act as a remote Flume client which sends Flume events via Avro to a Flume Agent configured with an Avro Source.
2. It can act as an embedded Flume Agent where Flume events pass directly into Flume for processing.
3. It can persist events to a local BerkeleyDB data store and then asynchronously send the events to Flume, similar to the embedded Flume Agent but without most of the Flume dependencies.

Usage as an embedded agent will cause the messages to be directly passed to the Flume Channel and then control will be immediately returned to the application. All interaction with remote agents will occur asynchronously. Setting the "type" attribute to "Embedded" will force the use of the embedded agent. In addition, configuring agent properties in the appender configuration will also cause the embedded agent to be used.

FlumeAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| agents | Agent[] | An array of Agents to which the logging events should be sent. If more than one agent is specified the first Agent will be the primary and subsequent Agents will be used in the order specified as secondaries should the primary Agent fail. Each Agent definition supplies the Agents host and port. The specification of agents and properties are mutually exclusive. If both are configured an error will result. |
| agentRetries | integer | The number of times the agent should be retried before failing to a secondary. This parameter is ignored when type="persistent" is specified (agents are tried once before failing to the next). |
| batchSize | integer | Specifies the number of events that should be sent as a batch. The default is 1. *This parameter only applies to the Flume Appender.* |
| compress | boolean | When set to true the message body will be compressed using gzip |
| connectTimeoutMillis | integer | The number of milliseconds Flume will wait before timing out the connection. |
| dataDir | String | Directory where the Flume write ahead log should be written. Valid only when embedded is set to true and Agent elements are used instead of Property elements. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| eventPrefix | String | The character string to prepend to each event attribute in order to distinguish it from MDC attributes. The default is an empty string. |
| flumeEventFactory | FlumeEventFactory | Factory that generates the Flume events from Log4j events. The default factory is the FlumeAvroAppender itself. |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is specified RFC5424Layout will be used. |
| lockTimeoutRetries | integer | The number of times to retry if a LockConflictException occurs while writing to Berkeley DB. The default is 5. |
| maxDelayMillis | integer | The maximum number of milliseconds to wait for batchSize events before publishing the batch. |

| | | |
|---|---|---|
| mdcExcludes | String | A comma separated list of mdc keys that should be excluded from the FlumeEvent. This is mutually exclusive with the mdcIncludes attribute. |
| mdcIncludes | String | A comma separated list of mdc keys that should be included in the FlumeEvent. Any keys in the MDC not found in the list will be excluded. This option is mutually exclusive with the mdcExcludes attribute. |
| mdcRequired | String | A comma separated list of mdc keys that must be present in the MDC. If a key is not present a LoggingException will be thrown. |
| mdcPrefix | String | A string that should be prepended to each MDC key in order to distinguish it from event attributes. The default string is "mdc:". |
| name | String | The name of the Appender. |
| properties | Property[] | One or more Property elements that are used to configure the Flume Agent. The properties must be configured without the agent name (the appender name is used for this) and no sources can be configured. Interceptors can be specified for the source using "sources.log4j-source.interceptors". All other Flume configuration properties are allowed. Specifying both Agent and Property elements will result in an error.<br>When used to configure in Persistent mode the valid properties are:<br>1. "keyProvider" to specify the name of the plugin to provide the secret key for encryption. |
| requestTimeoutMillis | integer | The number of milliseconds Flume will wait before timing out the request. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| type | enumeration | One of "Avro", "Embedded", or "Persistent" to indicate which variation of the Appender is desired. |

A sample FlumeAppender configuration that is configured with a primary and a secondary agent, compresses the body, and formats the body using the RFC5424Layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Flume name="eventLogger" compress="true">
      <Agent host="192.168.10.101" port="8800"/>
      <Agent host="192.168.10.102" port="8800"/>
      <RFC5424Layout enterpriseNumber="18060" includeMDC="true" appName="MyApp"/>
    </Flume>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="eventLogger"/>
    </Root>
  </Loggers>
</Configuration>
```

A sample FlumeAppender configuration that is configured with a primary and a secondary agent, compresses the body, formats the body using the RFC5424Layout, and persists encrypted events to disk:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Flume name="eventLogger" compress="true" type="persistent" dataDir="./logData">
      <Agent host="192.168.10.101" port="8800"/>
      <Agent host="192.168.10.102" port="8800"/>
      <RFC5424Layout enterpriseNumber="18060" includeMDC="true" appName="MyApp"/>
      <Property name="keyProvider">MySecretProvider</Property>
    </Flume>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="eventLogger"/>
    </Root>
  </Loggers>
</Configuration>
```

A sample FlumeAppender configuration that is configured with a primary and a secondary agent, compresses the body, formats the body using RFC5424Layout and passes the events to an embedded Flume Agent.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Flume name="eventLogger" compress="true" type="Embedded">
      <Agent host="192.168.10.101" port="8800"/>
      <Agent host="192.168.10.102" port="8800"/>
      <RFC5424Layout enterpriseNumber="18060" includeMDC="true" appName="MyApp"/>
    </Flume>
    <Console name="STDOUT">
      <PatternLayout pattern="%d [%p] %c %m%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Logger name="EventLogger" level="info">
      <AppenderRef ref="eventLogger"/>
    </Logger>
    <Root level="warn">
      <AppenderRef ref="STDOUT"/>
    </Root>
  </Loggers>
</Configuration>
```

A sample FlumeAppender configuration that is configured with a primary and a secondary agent using Flume configuration properties, compresses the body, formats the body using RFC5424Layout and passes the events to an embedded Flume Agent.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error" name="MyApp" packages="">
  <Appenders>
    <Flume name="eventLogger" compress="true" type="Embedded">
      <Property name="channels">file</Property>
      <Property name="channels.file.type">file</Property>
      <Property name="channels.file.checkpointDir">target/file-channel/checkpoint</Property>
      <Property name="channels.file.dataDirs">target/file-channel/data</Property>
      <Property name="sinks">agent1 agent2</Property>
      <Property name="sinks.agent1.channel">file</Property>
      <Property name="sinks.agent1.type">avro</Property>
      <Property name="sinks.agent1.hostname">192.168.10.101</Property>
      <Property name="sinks.agent1.port">8800</Property>
      <Property name="sinks.agent1.batch-size">100</Property>
      <Property name="sinks.agent2.channel">file</Property>
      <Property name="sinks.agent2.type">avro</Property>
      <Property name="sinks.agent2.hostname">192.168.10.102</Property>
      <Property name="sinks.agent2.port">8800</Property>
      <Property name="sinks.agent2.batch-size">100</Property>
      <Property name="sinkgroups">group1</Property>
      <Property name="sinkgroups.group1.sinks">agent1 agent2</Property>
      <Property name="sinkgroups.group1.processor.type">failover</Property>
      <Property name="sinkgroups.group1.processor.priority.agent1">10</Property>
      <Property name="sinkgroups.group1.processor.priority.agent2">5</Property>
      <RFC5424Layout enterpriseNumber="18060" includeMDC="true" appName="MyApp"/>
    </Flume>
    <Console name="STDOUT">
      <PatternLayout pattern="%d [%p] %c %m%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Logger name="EventLogger" level="info">
      <AppenderRef ref="eventLogger"/>
    </Logger>
    <Root level="warn">
      <AppenderRef ref="STDOUT"/>
    </Root>
  </Loggers>
</Configuration>
```

# JDBCAppender

The JDBCAppender writes log events to a relational database table using standard JDBC. It can be configured to obtain JDBC connections using a JNDI DataSource or a custom factory method. Whichever approach you take, it ***must*** be backed by a connection pool. Otherwise, logging performance will suffer greatly. If batch statements are supported by the configured JDBC driver and a bufferSize is configured to be a positive number, then log events will be batched. Note that as of Log4j 2.8, there are two ways to configure log event to column mappings: the original ColumnConfig style that only allows strings and timestamps, and the new ColumnMapping plugin that uses Log4j's built-in type conversion to allow for more data types (this is the same plugin as in the Cassandra Appender).

To get off the ground quickly during development, an alternative to using a connection source based on JNDI is to use the non-pooling DriverManager connection source. This connection source uses a JDBC connection string, a user name, and a password. Optionally, you can also use properties.

JDBCAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | *Required.* The name of the Appender. |

| | | | |
|---|---|---|---|
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. | |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. | |
| bufferSize | int | If an integer greater than 0, this causes the appender to buffer log events and flush whenever the buffer reaches this size. | |
| connectionSource | ConnectionSource | *Required.* The connections source from which database connections should be retrieved. | |
| tableName | String | *Required.* The name of the database table to insert log events into. | |
| columnConfigs | ColumnConfig[] | *Required (and/or columnMappings).* Information about the columns that log event data should be inserted into and how to insert that data. This is represented with multiple <Column> elements. | |
| columnMappings | ColumnMapping[] | *Required (and/or columnConfigs).* A list of column mapping configurations. Each column must specify a column name. Each column can have a conversion type specified by its fully qualified class name. By default, the conversion type is String. If the configured type is assignment-compatible with ReadOnlyStringMap (../log4j-api/apidocs/org/apache/logging/log4j/util/ReadOnlyStringMap.html) / ThreadContextMap (../log4j-api/apidocs/org/apache/logging/log4j/spi/ThreadContextMap.html) or ThreadContextStack (../log4j-api/apidocs/org/apache/logging/log4j/spi/ThreadContextStack.html), then that column will be populated with the MDC or NDC respectively (this is database-specific how they handle inserting a Map or List value). If the configured type is assignment-compatible with java.util.Date, then the log timestamp will be converted to that configured date type. If the configured type is assignment-compatible with java.sql.Clob or java.sql.NClob, then the formatted event will be set as a Clob or NClob respectively (similar to the traditional ColumnConfig plugin). If a literal attribute is given, then its value will be used as is in the INSERT query without any escaping. Otherwise, the layout or pattern specified will be converted into the configured type and stored in that column. | |
| immediateFail | boolean | false | When set to true, log events will not wait to try to reconnect and will fail immediately if the JDBC resources are not available. New in 2.11.2 |
| reconnectIntervalMillis | long | 5000 | If set to a value greater than 0, after an error, the JDBCDatabaseManager will attempt to reconnect to the database after waiting the specified number of milliseconds. If the reconnect fails then an exception will be thrown (which can be caught by the application if ignoreExceptions is set to false). New in 2.11.2. |

When configuring the JDBCAppender, you must specify a ConnectionSource implementation from which the Appender gets JDBC connections. You must use exactly one of the following nested elements:

- <DataSource>: Uses JNDI.
- <ConnectionFactory>: Points to a class-method pair to provide JDBC connections.
- <DriverManager>: A quick and dirty way to get off the ground, no connection pooling.
- <PoolingDriver>: Uses Apache Commons DBCP to provide connection pooling.

DataSource Parameters

| Parameter Name | Type | Description |
|---|---|---|
| jndiName | String | *Required.* The full, prefixed JNDI name that the javax.sql.DataSource is bound to, such as java:/comp/env/jdbc/LoggingDatabase. The DataSource must be backed by a connection pool; otherwise, logging will be very slow. |

ConnectionFactory Parameters

| Parameter Name | Type | Description |
|---|---|---|
| class | Class | *Required.* The fully qualified name of a class containing a static factory method for obtaining JDBC connections. |
| method | Method | *Required.* The name of a static factory method for obtaining JDBC connections. This method must have no parameters and its return type must be either java.sql.Connection or DataSource. If the method returns Connections, it must obtain them from a connection pool (and they will be returned to the pool when Log4j is done with them); otherwise, logging will be very slow. If the method returns a DataSource, the DataSource will only be retrieved once, and it must be backed by a connection pool for the same reasons. |

DriverManager Parameters

| Parameter Name | Type | Description |
|---|---|---|
| connectionString | String | *Required.* The driver-specific JDBC connection string. |
| userName | String | The database user name. You cannot specify both properties and a user name or password. |
| password | String | The database password. You cannot specify both properties and a user name or password. |
| driverClassName | String | The JDBC driver class name. Some old JDBC Driver can only be discovered by explicitly loading them by class name. |
| properties | Property[] | A list of properties. You cannot specify both properties and a user name or password. |

PoolingDriver Parameters (Apache Commons DBCP)

| Parameter Name | Type | Description |
|---|---|---|
| DriverManager parameters | DriverManager parameters | This connection source inherits all parameter from the DriverManager connection source. |
| poolName | String | The pool name used to pool JDBC Connections. Defaults to example. You can use the JDBC connection string prefix jdbc:apache:commons:dbcp: followed by the pool name if you want to use a pooled connection elsewhere. For example: jdbc:apache:commons:dbcp:example. |
| PoolableConnectionFactory | PoolableConnectionFactory element | Defines a PoolableConnectionFactory. |

PoolableConnectionFactory Parameters (Apache Commons DBCP)

| Parameter Name | Type | Description |
|---|---|---|
| autoCommitOnReturn | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| cacheState | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| ConnectionInitSqls | Strings | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| defaultAutoCommit | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| defaultCatalog | String | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| defaultQueryTimeoutSeconds | integer | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| defaultReadOnly | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| defaultTransactionIsolation | integer | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| disconnectionSqlCodes | Strings | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| fastFailValidation | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| maxConnLifetimeMillis | long | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| maxOpenPreparedStatements | integer | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| poolStatements | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| rollbackOnReturn | boolean | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| validationQuery | String | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |
| validationQueryTimeoutSeconds | integer | See Apache Commons DBCP PoolableConnectionFactory. (http://commons.apache.org/proper/commons-dbcp/api-2.4.0/org/apache/commons/dbcp2/PoolableConnectionFactory.html) |

When configuring the JDBCAppender, use the nested <Column> elements to specify which columns in the table should be written to and how to write to them. The JDBCAppender uses this information to formulate a PreparedStatement to insert records without SQL injection vulnerability.

Column Parameters

| Parameter Name | Type | Description |
|---|---|---|

| | | |
|---|---|---|
| name | String | *Required.* The name of the database column. |
| pattern | String | Use this attribute to insert a value or values from the log event in this column using a PatternLayout pattern. Simply specify any legal pattern in this attribute. Either this attribute, literal, or isEventTimestamp="true" must be specified, but not more than one of these. |
| literal | String | Use this attribute to insert a literal value in this column. The value will be included directly in the insert SQL, without any quoting (which means that if you want this to be a string, your value should contain single quotes around it like this: literal="'Literal String'"). This is especially useful for databases that don't support identity columns. For example, if you are using Oracle you could specify literal="NAME_OF_YOUR_SEQUENCE.NEXTVAL" to insert a unique ID in an ID column. Either this attribute, pattern, or isEventTimestamp="true" must be specified, but not more than one of these. |
| parameter | String | Use this attribute to insert an expression with a parameter marker '?' in this column. The value will be included directly in the insert SQL, without any quoting (which means that if you want this to be a string, your value should contain single quotes around it like this:<br><br><ColumnMapping name="instant" parameter="TIMESTAMPADD('MILLISECOND', ?, TIMESTAMP '1970-01-01')"/><br><br>You can only specify one of literal or parameter. |
| isEventTimestamp | boolean | Use this attribute to insert the event timestamp in this column, which should be a SQL datetime. The value will be inserted as a java.sql.Types.TIMESTAMP. Either this attribute (equal to true), pattern, or isEventTimestamp must be specified, but not more than one of these. |
| isUnicode | boolean | This attribute is ignored unless pattern is specified. If true or omitted (default), the value will be inserted as unicode (setNString or setNClob). Otherwise, the value will be inserted non-unicode (setString or setClob). |
| isClob | boolean | This attribute is ignored unless pattern is specified. Use this attribute to indicate that the column stores Character Large Objects (CLOBs). If true, the value will be inserted as a CLOB (setClob or setNClob). If false or omitted (default), the value will be inserted as a VARCHAR or NVARCHAR (setString or setNString). |

ColumnMapping Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | *Required.* The name of the database column. |
| pattern | String | Use this attribute to insert a value or values from the log event in this column using a PatternLayout pattern. Simply specify any legal pattern in this attribute. Either this attribute, literal, or isEventTimestamp="true" must be specified, but not more than one of these. |
| literal | String | Use this attribute to insert a literal value in this column. The value will be included directly in the insert SQL, without any quoting (which means that if you want this to be a string, your value should contain single quotes around it like this: literal="'Literal String'"). This is especially useful for databases that don't support identity columns. For example, if you are using Oracle you could specify literal="NAME_OF_YOUR_SEQUENCE.NEXTVAL" to insert a unique ID in an ID column. Either this attribute, pattern, or isEventTimestamp="true" must be specified, but not more than one of these. |
| layout | Layout | The Layout to format the LogEvent. |
| type | String | Conversion type name, a fully-qualified class name. |

Here are a couple sample configurations for the JDBCAppender, as well as a sample factory implementation that uses Commons Pooling and Commons DBCP to pool database connections:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <JDBC name="databaseAppender" tableName="dbo.application_log">
      <DataSource jndiName="java:/comp/env/jdbc/LoggingDataSource" />
      <Column name="eventDate" isEventTimestamp="true" />
      <Column name="level" pattern="%level" />
      <Column name="logger" pattern="%logger" />
      <Column name="message" pattern="%message" />
      <Column name="exception" pattern="%ex{full}" />
    </JDBC>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <JDBC name="databaseAppender" tableName="LOGGING.APPLICATION_LOG">
      <ConnectionFactory class="net.example.db.ConnectionFactory" method="getDatabaseConnection" />
      <Column name="EVENT_ID" literal="LOGGING.APPLICATION_LOG_SEQUENCE.NEXTVAL" />
      <Column name="EVENT_DATE" isEventTimestamp="true" />
      <Column name="LEVEL" pattern="%level" />
      <Column name="LOGGER" pattern="%logger" />
      <Column name="MESSAGE" pattern="%message" />
      <Column name="THROWABLE" pattern="%ex{full}" />
    </JDBC>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

```
package net.example.db;

import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

import javax.sql.DataSource;

import org.apache.commons.dbcp.DriverManagerConnectionFactory;
import org.apache.commons.dbcp.PoolableConnection;
import org.apache.commons.dbcp.PoolableConnectionFactory;
import org.apache.commons.dbcp.PoolingDataSource;
import org.apache.commons.pool.impl.GenericObjectPool;

public class ConnectionFactory {
    private static interface Singleton {
        final ConnectionFactory INSTANCE = new ConnectionFactory();
    }

    private final DataSource dataSource;

    private ConnectionFactory() {
        Properties properties = new Properties();
        properties.setProperty("user", "logging");
        properties.setProperty("password", "abc123"); // or get properties from some configuration file

        GenericObjectPool<PoolableConnection> pool = new GenericObjectPool<PoolableConnection>();
        DriverManagerConnectionFactory connectionFactory = new DriverManagerConnectionFactory(
                "jdbc:mysql://example.org:3306/exampleDb", properties
        );
        new PoolableConnectionFactory(
                connectionFactory, pool, null, "SELECT 1", 3, false, false, Connection.TRANSACTION_READ_COMMITTED
        );

        this.dataSource = new PoolingDataSource(pool);
    }

    public static Connection getDatabaseConnection() throws SQLException {
        return Singleton.INSTANCE.dataSource.getConnection();
    }
}
```

This appender is MapMessage (messages.html#MapMessage)-aware.

The following configuration uses a MessageLayout to indicate that the Appender should match the keys of a MapMessage to the names of ColumnMappings when setting the values of the Appender's SQL INSERT statement. This let you insert rows for custom values in a database table based on a Log4j MapMessage instead of values from LogEvents.

```
<Configuration status="debug">

  <Appenders>
    <Console name="STDOUT">
      <PatternLayout pattern="%C{1.} %m %level MDC%X%n"/>
    </Console>
    <Jdbc name="databaseAppender" tableName="dsLogEntry" ignoreExceptions="false">
      <DataSource jndiName="java:/comp/env/jdbc/TestDataSourceAppender" />
      <ColumnMapping name="Id" />
      <ColumnMapping name="ColumnA" />
      <ColumnMapping name="ColumnB" />
      <MessageLayout />
    </Jdbc>
  </Appenders>

  <Loggers>
    <Logger name="org.apache.logging.log4j.core.appender.db" level="debug" additivity="false">
      <AppenderRef ref="databaseAppender" />
    </Logger>

    <Root level="fatal">
      <AppenderRef ref="STDOUT"/>
    </Root>
  </Loggers>

</Configuration>
```

# JMS Appender

The JMS Appender sends the formatted log event to a JMS Destination.

Note that in Log4j 2.0, this appender was split into a JMSQueueAppender and a JMSTopicAppender. Starting in Log4j 2.1, these appenders were combined into the JMS Appender which makes no distinction between queues and topics. However, configurations written for 2.0 which use the <JMSQueue/> or <JMSTopic/> elements will continue to work with the new <JMS/> configuration element.

JMS Appender Parameters

| Parameter Name | Type | Default | Description |
|---|---|---|---|
| factoryBindingName | String | *Required* | The name to locate in the Context that provides the ConnectionFactory (http://download.oracle.com/javaee/5/api/javax/jms/ConnectionFactory.html). This can be any subinterface of ConnectionFactory as well. |
| factoryName | String | *Required* | The fully qualified class name that should be used to define the Initial Context Factory as defined in INITIAL_CONTEXT_FACTORY (http://download.oracle.com/javase/7/docs/api/javax/naming/Context.html#INITIAL_CONTEXT_FACTORY). If a factoryName is specified without a providerURL a warning message will be logged as this is likely to cause problems. |
| filter | Filter | null | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| layout | Layout | *Required* | The Layout to use to format the LogEvent. *New since 2.9, in previous versions SerializedLayout was default.* |
| name | String | *Required* | The name of the Appender. |
| password | String | null | The password to use to create the JMS connection. |
| providerURL | String | *Required* | The URL of the provider to use as defined by PROVIDER_URL (http://download.oracle.com/javase/7/docs/api/javax/naming/Context.html#PROVIDER_URL). |
| destinationBindingName | String | *Required* | The name to use to locate the Destination (http://download.oracle.com/javaee/5/api/javax/jms/Destination.html). This can be a Queue or Topic, and as such, the attribute names queueBindingName and topicBindingName are aliases to maintain compatibility with the Log4j 2.0 JMS appenders. |
| securityPrincipalName | String | null | The name of the identity of the Principal as specified by SECURITY_PRINCIPAL (http://download.oracle.com/javase/7/docs/api/javax/naming/Context.html#SECURITY_PRINCIPAL). If a securityPrincipalName is specified without securityCredentials a warning message will be logged as this is likely to cause problems. |
| securityCredentials | String | null | The security credentials for the principal as specified by SECURITY_CREDENTIALS (http://download.oracle.com/javase/7/docs/api/javax/naming/Context.html#SECURITY_CREDENTIALS). |
| ignoreExceptions | boolean | true | When true, exceptions caught while appending events are internally logged and then ignored. When false exceptions are propagated to the caller. You must set this to false when wrapping this Appender in a FailoverAppender. |
| immediateFail | boolean | false | When set to true, log events will not wait to try to reconnect and will fail immediately if the JMS resources are not available. New in 2.9. |
| reconnectIntervalMillis | long | 5000 | If set to a value greater than 0, after an error, the JMSManager will attempt to reconnect to the broker after waiting the specified number of milliseconds. If the reconnect fails then an exception will be thrown (which can be caught by the application if ignoreExceptions is set to false). New in 2.9. |

| | | | |
|---|---|---|---|
| urlPkgPrefixes | String | null | A colon-separated list of package prefixes for the class name of the factory class that will create a URL context factory as defined by URL_PKG_PREFIXES (http://download.oracle.com/javase/7/docs/api/javax/naming/Context.html#URL_PKG_PREFIXES). |
| userName | String | null | The user id used to create the JMS connection. |

Here is a sample JMS Appender configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp">
  <Appenders>
    <JMS name="jmsQueue" destinationBindingName="MyQueue"
         factoryBindingName="MyQueueConnectionFactory">
      <JsonLayout properties="true"/>
    </JMS>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="jmsQueue"/>
    </Root>
  </Loggers>
</Configuration>
```

To map your Log4j MapMessages to JMS javax.jms.MapMessages, set the layout of the appender to MessageLayout with <MessageLayout /> (Since 2.9.):

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp">
  <Appenders>
    <JMS name="jmsQueue" destinationBindingName="MyQueue"
         factoryBindingName="MyQueueConnectionFactory">
      <MessageLayout />
    </JMS>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="jmsQueue"/>
    </Root>
  </Loggers>
</Configuration>
```

# JPAAppender

As of Log4j 2.11.0, JPA support has moved from the existing module log4j-core to the new module log4j-jpa.

The JPAAppender writes log events to a relational database table using the Java Persistence API 2.1. It requires the API and a provider implementation be on the classpath. It also requires a decorated entity configured to persist to the table desired. The entity should either extend org.apache.logging.log4j.core.appender.db.jpa.BasicLogEventEntity (if you mostly want to use the default mappings) and provide at least an @Id property, or org.apache.logging.log4j.core.appender.db.jpa.AbstractLogEventWrapperEntity (if you want to significantly customize the mappings). See the Javadoc for these two classes for more information. You can also consult the source code of these two classes as an example of how to implement the entity.

JPAAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | *Required.* The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| bufferSize | int | If an integer greater than 0, this causes the appender to buffer log events and flush whenever the buffer reaches this size. |
| entityClassName | String | *Required.* The fully qualified name of the concrete LogEventWrapperEntity implementation that has JPA annotations mapping it to a database table. |
| persistenceUnitName | String | *Required.* The name of the JPA persistence unit that should be used for persisting log events. |

Here is a sample configuration for the JPAAppender. The first XML sample is the Log4j configuration file, the second is the persistence.xml file. EclipseLink is assumed here, but any JPA 2.1 or higher provider will do. You should *always* create a *separate* persistence unit for logging, for two reasons. First, <shared-cache-mode> *must* be set to "NONE," which is usually not desired in normal JPA usage. Also, for performance reasons the logging entity should be isolated in its own persistence unit away from all other entities and you should use a non-JTA data source. Note that your persistence unit *must* also contain <class> elements for all of the org.apache.logging.log4j.core.appender.db.jpa.converter converter classes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <JPA name="databaseAppender" persistenceUnitName="loggingPersistenceUnit"
         entityClassName="com.example.logging.JpaLogEntity" />
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
             version="2.1">

  <persistence-unit name="loggingPersistenceUnit" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.ContextMapAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.ContextMapJsonAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.ContextStackAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.ContextStackJsonAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.MarkerAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.MessageAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.StackTraceElementAttributeConverter</class>
    <class>org.apache.logging.log4j.core.appender.db.jpa.converter.ThrowableAttributeConverter</class>
    <class>com.example.logging.JpaLogEntity</class>
    <non-jta-data-source>jdbc/LoggingDataSource</non-jta-data-source>
    <shared-cache-mode>NONE</shared-cache-mode>
  </persistence-unit>

</persistence>
```

```java
package com.example.logging;
...
@Entity
@Table(name="application_log", schema="dbo")
public class JpaLogEntity extends BasicLogEventEntity {
    private static final long serialVersionUID = 1L;
    private long id = 0L;

    public TestEntity() {
        super(null);
    }
    public TestEntity(LogEvent wrappedEvent) {
        super(wrappedEvent);
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    public long getId() {
        return this.id;
    }

    public void setId(long id) {
        this.id = id;
    }

    // If you want to override the mapping of any properties mapped in BasicLogEventEntity,
    // just override the getters and re-specify the annotations.
}
```

```java
package com.example.logging;
...
@Entity
@Table(name="application_log", schema="dbo")
public class JpaLogEntity extends AbstractLogEventWrapperEntity {
    private static final long serialVersionUID = 1L;
    private long id = 0L;

    public TestEntity() {
        super(null);
    }
    public TestEntity(LogEvent wrappedEvent) {
        super(wrappedEvent);
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "logEventId")
    public long getId() {
        return this.id;
    }

    public void setId(long id) {
        this.id = id;
    }

    @Override
    @Enumerated(EnumType.STRING)
    @Column(name = "level")
    public Level getLevel() {
        return this.getWrappedEvent().getLevel();
    }

    @Override
    @Column(name = "logger")
    public String getLoggerName() {
        return this.getWrappedEvent().getLoggerName();
    }

    @Override
    @Column(name = "message")
    @Convert(converter = MyMessageConverter.class)
    public Message getMessage() {
        return this.getWrappedEvent().getMessage();
    }
    ...
}
```

# HttpAppender

The HttpAppender sends log events over HTTP. A Layout must be provided to format the LogEvent.

Will set the Content-Type header according to the layout. Additional headers can be specified with embedded Property elements.

Will wait for response from server, and throw error if no 2xx response is received.

Implemented with HttpURLConnection (https://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html) .

HttpAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | The name of the Appender. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| layout | Layout | The Layout to use to format the LogEvent. |
| Ssl | SslConfiguration | Contains the configuration for the KeyStore and TrustStore for https. Optional, uses Java runtime defaults if not specified. See SSL |
| verifyHostname | boolean | Whether to verify server hostname against certificate. Only valid for https. Optional, defaults to true |
| url | string | The URL to use. The URL scheme must be "http" or "https". |
| method | string | The HTTP method to use. Optional, default is "POST". |
| connectTimeoutMillis | integer | The connect timeout in milliseconds. Optional, default is 0 (infinite timeout). |
| readTimeoutMillis | integer | The socket read timeout in milliseconds. Optional, default is 0 (infinite timeout). |
| headers | Property[] | Additional HTTP headers to use. The values support lookups (lookups.html). |

| | | |
|---|---|---|
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |

Here is a sample HttpAppender configuration snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
  ...
  <Appenders>
    <Http name="Http" url="https://localhost:9200/test/log4j/">
      <Property name="X-Java-Runtime" value="$${java:runtime}" />
      <JsonLayout properties="true"/>
      <SSL>
        <KeyStore   location="log4j2-keystore.jks" passwordEnvironmentVariable="KEYSTORE_PASSWORD"/>
        <TrustStore location="truststore.jks"      passwordFile="${sys:user.home}/truststore.pwd"/>
      </SSL>
    </Http>
  </Appenders>
```

# KafkaAppender

The KafkaAppender logs events to an Apache Kafka (https://kafka.apache.org/) topic. Each log event is sent as a Kafka record.

KafkaAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| topic | String | The Kafka topic to use. Required. |
| key | String | The key that will be sent to Kafka with every message. Optional value defaulting to null. Any of the Lookups (./lookups.html)) can be included. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| layout | Layout | The Layout to use to format the LogEvent. Required, there is no default. *New since 2.9, in previous versions <PatternLayout pattern="%m"/> was default.* |
| name | String | The name of the Appender. Required. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| syncSend | boolean | The default is true, causing sends to block until the record has been acknowledged by the Kafka server. When set to false sends return immediately, allowing for lower latency and significantly higher throughput. *New since 2.8. Be aware that this is a new addition, and it has not been extensively tested. Any failure sending to Kafka will be reported as error to StatusLogger and the log event will be dropped (the ignoreExceptions parameter will not be effective). Log events may arrive out of order to the Kafka server.* |
| properties | Property[] | You can set properties in Kafka producer properties (http://kafka.apache.org/documentation.html#producerconfigs) . You need to set the bootstrap.servers property, there are sensible default values for the others. Do not set the value.serializer nor key.serializer properties. |

Here is a sample KafkaAppender configuration snippet:

```
<?xml version="1.0" encoding="UTF-8"?>
  ...
  <Appenders>
    <Kafka name="Kafka" topic="log-test">
      <PatternLayout pattern="%date %message"/>
        <Property name="bootstrap.servers">localhost:9092</Property>
    </Kafka>
  </Appenders>
```

This appender is synchronous by default and will block until the record has been acknowledged by the Kafka server, timeout for this can be set with the timeout.ms property (defaults to 30 seconds). Wrap with Async appender (http://logging.apache.org/log4j/2.x/manual/appenders.html#AsyncAppender) and/or set syncSend to false to log asynchronously.

This appender requires the Kafka client library (http://kafka.apache.org/) . Note that you need to use a version of the Kafka client library matching the Kafka server used.

*Note*:Make sure to not let org.apache.kafka log to a Kafka appender on DEBUG level, since that will cause recursive logging:

```
<?xml version="1.0" encoding="UTF-8"?>
  ...
  <Loggers>
    <Root level="DEBUG">
      <AppenderRef ref="Kafka"/>
    </Root>
    <Logger name="org.apache.kafka" level="INFO" /> <!-- avoid recursive logging -->
  </Loggers>
```

# MemoryMappedFileAppender

*New since 2.1. Be aware that this is a new addition, and although it has been tested on several platforms, it does not have as much track record as the other file appenders.*

The MemoryMappedFileAppender maps a part of the specified file into memory and writes log events to this memory, relying on the operating system's virtual memory manager to synchronize the changes to the storage device. The main benefit of using memory mapped files is I/O performance. Instead of making system calls to write to disk, this appender can simply change the program's local memory, which is orders of magnitude faster. Also, in most operating systems the memory region mapped actually is the kernel's page cache (http://en.wikipedia.org/wiki/Page_cache) (file cache), meaning that no copies need to be created in user space. (TODO: performance tests that compare performance of this appender to RandomAccessFileAppender and FileAppender.)

There is some overhead with mapping a file region into memory, especially very large regions (half a gigabyte or more). The default region size is 32 MB, which should strike a reasonable balance between the frequency and the duration of remap operations. (TODO: performance test remapping various sizes.)

Similar to the FileAppender and the RandomAccessFileAppender, MemoryMappedFileAppender uses a MemoryMappedFileManager to actually perform the file I/O. While MemoryMappedFileAppender from different Configurations cannot be shared, the MemoryMappedFileManagers can be if the Manager is accessible. For example, two web applications in a servlet container can have their own configuration and safely write to the same file if Log4j is in a ClassLoader that is common to both of them.

MemoryMappedFileAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| append | boolean | When true - the default, records will be appended to the end of the file. When set to false, the file will be cleared before new records are written. |
| fileName | String | The name of the file to write to. If the file, or any of its parent directories, do not exist, they will be created. |
| filters | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| immediateFlush | boolean | When set to true, each write will be followed by a call to MappedByteBuffer.force() (http://docs.oracle.com/javase/7/docs/api/java/nio/MappedByteBuffer.html#force()) . This will guarantee the data is written to the storage device.<br>The default for this parameter is false. This means that the data is written to the storage device even if the Java process crashes, but there may be data loss if the operating system crashes.<br>Note that manually forcing a sync on every log event loses most of the performance benefits of using a memory mapped file.<br>Flushing after every write is only useful when using this appender with synchronous loggers. Asynchronous loggers and appenders will automatically flush at the end of a batch of events, even if immediateFlush is set to false. This also guarantees the data is written to disk but is more efficient. |
| regionLength | int | The length of the mapped region, defaults to 32 MB (32 * 1024 * 1024 bytes). This parameter must be a value between 256 and 1,073,741,824 (1 GB or 2^30); values outside this range will be adjusted to the closest valid value. Log4j will round the specified value up to the nearest power of two. |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| name | String | The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |

Here is a sample MemoryMappedFile configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <MemoryMappedFile name="MyFile" fileName="logs/app.log">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
    </MemoryMappedFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="MyFile"/>
    </Root>
  </Loggers>
</Configuration>
```

# NoSQLAppender

The NoSQLAppender writes log events to a NoSQL database using an internal lightweight provider interface. Provider implementations currently exist for MongoDB and Apache CouchDB, and writing a custom provider is quite simple.

NoSQLAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | *Required.* The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |

| | | |
|---|---|---|
| bufferSize | int | If an integer greater than 0, this causes the appender to buffer log events and flush whenever the buffer reaches this size. |
| NoSqlProvider | NoSQLProvider<C extends NoSQLConnection<W, T extends NoSQLObject<W>>> | *Required.* The NoSQL provider that provides connections to the chosen NoSQL database. |

You specify which NoSQL provider to use by specifying the appropriate configuration element within the <NoSql> element. The types currently supported are <MongoDb> and <CouchDb>. To create your own custom provider, read the JavaDoc for the NoSQLProvider, NoSQLConnection, and NoSQLObject classes and the documentation about creating Log4j plugins. We recommend you review the source code for the MongoDB and CouchDB providers as a guide for creating your own provider.

The following example demonstrates how log events are persisted in NoSQL databases if represented in a JSON format:

```
{
    "level": "WARN",
    "loggerName": "com.example.application.MyClass",
    "message": "Something happened that you might want to know about.",
    "source": {
        "className": "com.example.application.MyClass",
        "methodName": "exampleMethod",
        "fileName": "MyClass.java",
        "lineNumber": 81
    },
    "marker": {
        "name": "SomeMarker",
        "parent" {
            "name": "SomeParentMarker"
        }
    },
    "threadName": "Thread-1",
    "millis": 1368844166761,
    "date": "2013-05-18T02:29:26.761Z",
    "thrown": {
        "type": "java.sql.SQLException",
        "message": "Could not insert record. Connection lost.",
        "stackTrace": [
                { "className": "org.example.sql.driver.PreparedStatement$1", "methodName": "responder", "fileName": "PreparedStatement.java", "lineNumbe
r": 1049 },
                { "className": "org.example.sql.driver.PreparedStatement", "methodName": "executeUpdate", "fileName": "PreparedStatement.java", "lineNum
ber": 738 },
                { "className": "com.example.application.MyClass", "methodName": "exampleMethod", "fileName": "MyClass.java", "lineNumber": 81 },
                { "className": "com.example.application.MainClass", "methodName": "main", "fileName": "MainClass.java", "lineNumber": 52 }
        ],
        "cause": {
            "type": "java.io.IOException",
            "message": "Connection lost.",
            "stackTrace": [
                { "className": "java.nio.channels.SocketChannel", "methodName": "write", "fileName": null, "lineNumber": -1 },
                { "className": "org.example.sql.driver.PreparedStatement$1", "methodName": "responder", "fileName": "PreparedStatement.java", "lineNumbe
r": 1032 },
                { "className": "org.example.sql.driver.PreparedStatement", "methodName": "executeUpdate", "fileName": "PreparedStatement.java", "lineNum
ber": 738 },
                { "className": "com.example.application.MyClass", "methodName": "exampleMethod", "fileName": "MyClass.java", "lineNumber": 81 },
                { "className": "com.example.application.MainClass", "methodName": "main", "fileName": "MainClass.java", "lineNumber": 52 }
            ]
        }
    },
    "contextMap": {
        "ID": "86c3a497-4e67-4eed-9d6a-2e5797324d7b",
        "username": "JohnDoe"
    },
    "contextStack": [
        "topItem",
        "anotherItem",
        "bottomItem"
    ]
}
```

# NoSQLAppender for MongoDB

Starting with Log4 2.11.0, we provide two MongoDB modules:

- log4j-mongodb2 defines the configuration element MongoDb2 matching the MongoDB Driver version 2.
- log4j-mongodb3 defines the configuration element MongoDb3 matching the MongoDB Driver version 3.

We no longer provide the module log4j-mongodb.

The module log4j-mongodb2 aliases the old configuration element MongoDb to MongoDb2.

# NoSQLAppender for MongoDB 2

This section details specializations of the NoSQLAppender provider for MongoDB using the MongoDB driver version 2. The NoSQLAppender Appender writes log events to a NoSQL database using an internal lightweight provider interface.

MongoDB2 Provider Parameters

| Parameter Name | Type | Description |
|---|---|---|
| collectionName | String | *Required.* The name of the MongoDB collection to insert the events into. |
| writeConcernConstant | Field | By default, the MongoDB provider inserts records with the instructions com.mongodb.WriteConcern.ACKNOWLEDGED. Use this optional attribute to specify the name of a constant other than ACKNOWLEDGED. |
| writeConcernConstantClass | Class | If you specify writeConcernConstant, you can use this attribute to specify a class other than com.mongodb.WriteConcern to find the constant on (to create your own custom instructions). |
| factoryClassName | Class | To provide a connection to the MongoDB database, you can use this attribute and factoryMethodName to specify a class and static method to get the connection from. The method must return a com.mongodb.DB or a com.mongodb.MongoClient. If the DB is not authenticated, you must also specify a username and password. If you use the factory method for providing a connection, you must not specify the databaseName, server, or port attributes. |
| factoryMethodName | Method | See the documentation for attribute factoryClassName. |
| databaseName | String | If you do not specify a factoryClassName and factoryMethodName for providing a MongoDB connection, you must specify a MongoDB database name using this attribute. You must also specify a username and password. You can optionally also specify a server (defaults to localhost), and a port (defaults to the default MongoDB port). |
| server | String | See the documentation for attribute databaseName. |
| port | int | See the documentation for attribute databaseName. |
| username | String | See the documentation for attributes databaseName and factoryClassName. |
| password | String | See the documentation for attributes databaseName and factoryClassName. |
| capped | boolean | Enable support for capped collections (https://docs.mongodb.com/manual/core/capped-collections/) |
| collectionSize | int | Specify the size in bytes of the capped collection to use if enabled. The minimum size is 4096 bytes, and larger sizes will be increased to the nearest integer multiple of 256. See the capped collection documentation linked above for more information. |

This appender is MapMessage (messages.html#MapMessage)-aware.

Here are a few sample configurations for the NoSQLAppender and MongoDB2 provider:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <NoSql name="databaseAppender">
      <MongoDb2 databaseName="applicationDb" collectionName="applicationLog" server="mongo.example.org"
                username="loggingUser" password="abc123" />
    </NoSql>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <NoSql name="databaseAppender">
      <MongoDb2 collectionName="applicationLog" factoryClassName="org.example.db.ConnectionFactory"
                factoryMethodName="getNewMongoClient" />
    </NoSql>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

Starting in Log4j version 2.11.0, the provider element name is MongoDb2. The name MongoDb is now a deprecated alias for MongoDb2.

# NoSQLAppender for MongoDB 3

This section details specializations of the NoSQLAppender provider for MongoDB using the MongoDB driver version 3. The NoSQLAppender Appender writes log events to a NoSQL database using an internal lightweight provider interface.

MongoDB3 Provider Parameters

| Parameter Name | Type | Description |
|---|---|---|
| collectionName | String | *Required.* The name of the MongoDB collection to insert the events into. |

| | | |
|---|---|---|
| writeConcernConstant | Field | By default, the MongoDB provider inserts records with the instructions com.mongodb.WriteConcern.ACKNOWLEDGED. Use this optional attribute to specify the name of a constant other than ACKNOWLEDGED. |
| writeConcernConstantClass | Class | If you specify writeConcernConstant, you can use this attribute to specify a class other than com.mongodb.WriteConcern to find the constant on (to create your own custom instructions). |
| factoryClassName | Class | To provide a connection to the MongoDB database, you can use this attribute and factoryMethodName to specify a class and static method to get the connection from. The method must return a com.mongodb.client.MongoDatabase or a com.mongodb.MongoClient. If the com.mongodb.client.MongoDatabase is not authenticated, you must also specify a username and password. If you use the factory method for providing a connection, you must not specify the databaseName, server, or port attributes. |
| factoryMethodName | Method | See the documentation for attribute factoryClassName. |
| databaseName | String | If you do not specify a factoryClassName and factoryMethodName for providing a MongoDB connection, you must specify a MongoDB database name using this attribute. You must also specify a username and password. You can optionally also specify a server (defaults to localhost), and a port (defaults to the default MongoDB port). |
| server | String | See the documentation for attribute databaseName. |
| port | int | See the documentation for attribute databaseName. |
| username | String | See the documentation for attributes databaseName and factoryClassName. |
| password | String | See the documentation for attributes databaseName and factoryClassName. |
| capped | boolean | Enable support for capped collections (https://docs.mongodb.com/manual/core/capped-collections/) |
| collectionSize | int | Specify the size in bytes of the capped collection to use if enabled. The minimum size is 4096 bytes, and larger sizes will be increased to the nearest integer multiple of 256. See the capped collection documentation linked above for more information. |

This appender is MapMessage (messages.html#MapMessage)-aware.

Here are a few sample configurations for the NoSQLAppender and MongoDB3 provider:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <NoSql name="databaseAppender">
      <MongoDb3 databaseName="applicationDb" collectionName="applicationLog" server="mongo.example.org"
               username="loggingUser" password="abc123" />
    </NoSql>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <NoSql name="databaseAppender">
      <MongoDb3 collectionName="applicationLog" factoryClassName="org.example.db.ConnectionFactory"
               factoryMethodName="getNewMongoClient" />
    </NoSql>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

## NoSQLAppender for Apache CouchDB

This section details specializations of the NoSQLAppender provider for CouchDB. The NoSQLAppender writes log events to a NoSQL database using an internal lightweight provider interface.

CouchDB Provider Parameters

| Parameter Name | Type | Description |
|---|---|---|
| factoryClassName | Class | To provide a connection to the CouchDB database, you can use this attribute and factoryMethodName to specify a class and static method to get the connection from. The method must return a org.lightcouch.CouchDbClient or a org.lightcouch.CouchDbProperties. If you use the factory method for providing a connection, you must not specify the databaseName, protocol, server, port, username, or password attributes. |
| factoryMethodName | Method | See the documentation for attribute factoryClassName. |
| databaseName | String | If you do not specify a factoryClassName and factoryMethodName for providing a CouchDB connection, you must specify a CouchDB database name using this attribute. You must also specify a username and password. You can optionally also specify a protocol (defaults to http), server (defaults to localhost), and a port (defaults to 80 for http and 443 for https). |

| | | |
|---|---|---|
| protocol | String | Must either be "http" or "https." See the documentation for attribute databaseName. |
| server | String | See the documentation for attribute databaseName. |
| port | int | See the documentation for attribute databaseName. |
| username | String | See the documentation for attributes databaseName. |
| password | String | See the documentation for attributes databaseName. |

Here are a few sample configurations for the NoSQLAppender and CouchDB provider:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="error">
  <Appenders>
    <NoSql name="databaseAppender">
      <CouchDb databaseName="applicationDb" protocol="https" server="couch.example.org"
               username="loggingUser" password="abc123" />
    </NoSql>
  </Appenders>
  <Loggers>
    <Root level="warn">
      <AppenderRef ref="databaseAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

# OutputStreamAppender

The OutputStreamAppender provides the base for many of the other Appenders such as the File and Socket appenders that write the event to an Output Stream. It cannot be directly configured. Support for immediateFlush and buffering is provided by the OutputStreamAppender. The OutputStreamAppender uses an OutputStreamManager to handle the actual I/O, allowing the stream to be shared by Appenders in multiple configurations.

# RandomAccessFileAppender

The RandomAccessFileAppender is similar to the standard FileAppender except it is always buffered (this cannot be switched off) and internally it uses a ByteBuffer + RandomAccessFile instead of a BufferedOutputStream. We saw a 20-200% performance improvement compared to FileAppender with "bufferedIO=true" in our measurements (../performance.html#whichAppender). Similar to the FileAppender, RandomAccessFileAppender uses a RandomAccessFileManager to actually perform the file I/O. While RandomAccessFileAppender from different Configurations cannot be shared, the RandomAccessFileManagers can be if the Manager is accessible. For example, two web applications in a servlet container can have their own configuration and safely write to the same file if Log4j is in a ClassLoader that is common to both of them.

RandomAccessFileAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| append | boolean | When true - the default, records will be appended to the end of the file. When set to false, the file will be cleared before new records are written. |
| fileName | String | The name of the file to write to. If the file, or any of its parent directories, do not exist, they will be created. |
| filters | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| immediateFlush | boolean | When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance.<br>Flushing after every write is only useful when using this appender with synchronous loggers. Asynchronous loggers and appenders will automatically flush at the end of a batch of events, even if immediateFlush is set to false. This also guarantees the data is written to disk but is more efficient. |
| bufferSize | int | The buffer size, defaults to 262,144 bytes (256 * 1024). |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| name | String | The name of the Appender. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |

Here is a sample RandomAccessFile configuration:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RandomAccessFile name="MyFile" fileName="logs/app.log">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
    </RandomAccessFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="MyFile"/>
    </Root>
  </Loggers>
</Configuration>
```

# RewriteAppender

The RewriteAppender allows the LogEvent to manipulated before it is processed by another Appender. This can be used to mask sensitive information such as passwords or to inject information into each event. The RewriteAppender must be configured with a RewritePolicy (RewritePolicy). The RewriteAppender should be configured after any Appenders it references to allow it to shut down properly.

RewriteAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| AppenderRef | String | The name of the Appenders to call after the LogEvent has been manipulated. Multiple AppenderRef elements can be configured. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| name | String | The name of the Appender. |
| rewritePolicy | RewritePolicy | The RewritePolicy that will manipulate the LogEvent. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |

## RewritePolicy

RewritePolicy is an interface that allows implementations to inspect and possibly modify LogEvents before they are passed to Appender. RewritePolicy declares a single method named rewrite that must be implemented. The method is passed the LogEvent and can return the same event or create a new one.

#### MapRewritePolicy

MapRewritePolicy will evaluate LogEvents that contain a MapMessage and will add or update elements of the Map.

| Parameter Name | Type | Description |
|---|---|---|
| mode | String | "Add" or "Update" |
| keyValuePair | KeyValuePair[] | An array of keys and their values. |

The following configuration shows a RewriteAppender configured to add a product key and its value to the MapMessage.:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Console name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="%m%n"/>
    </Console>
    <Rewrite name="rewrite">
      <AppenderRef ref="STDOUT"/>
      <MapRewritePolicy mode="Add">
        <KeyValuePair key="product" value="TestProduct"/>
      </MapRewritePolicy>
    </Rewrite>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Rewrite"/>
    </Root>
  </Loggers>
</Configuration>
```

#### PropertiesRewritePolicy

PropertiesRewritePolicy will add properties configured on the policy to the ThreadContext Map being logged. The properties will not be added to the actual ThreadContext Map. The property values may contain variables that will be evaluated when the configuration is processed as well as when the event is logged.

| Parameter Name | Type | Description |
|---|---|---|
| properties | Property[] | One of more Property elements to define the keys and values to be added to the ThreadContext Map. |

The following configuration shows a RewriteAppender configured to add a product key and its value to the MapMessage:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Console name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="%m%n"/>
    </Console>
    <Rewrite name="rewrite">
      <AppenderRef ref="STDOUT"/>
      <PropertiesRewritePolicy>
        <Property name="user">${sys:user.name}</Property>
        <Property name="env">${sys:environment}</Property>
      </PropertiesRewritePolicy>
    </Rewrite>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Rewrite"/>
    </Root>
  </Loggers>
</Configuration>
```

#### LoggerNameLevelRewritePolicy

You can use this policy to make loggers in third party code less chatty by changing event levels. The LoggerNameLevelRewritePolicy will rewrite log event levels for a given logger name prefix. You configure a LoggerNameLevelRewritePolicy with a logger name prefix and a pairs of levels, where a pair defines a source level and a target level.

| Parameter Name | Type | Description |
|---|---|---|
| logger | String | A logger name used as a prefix to test each event's logger name. |
| LevelPair | KeyValuePair[] | An array of keys and their values, each key is a source level, each value a target level. |

The following configuration shows a RewriteAppender configured to map level INFO to DEBUG and level WARN to INFO for all loggers that start with com.foo.bar.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp">
  <Appenders>
    <Console name="STDOUT" target="SYSTEM_OUT">
      <PatternLayout pattern="%m%n"/>
    </Console>
    <Rewrite name="rewrite">
      <AppenderRef ref="STDOUT"/>
      <LoggerNameLevelRewritePolicy logger="com.foo.bar">
        <KeyValuePair key="INFO" value="DEBUG"/>
        <KeyValuePair key="WARN" value="INFO"/>
      </LoggerNameLevelRewritePolicy>
    </Rewrite>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Rewrite"/>
    </Root>
  </Loggers>
</Configuration>
```

## RollingFileAppender

The RollingFileAppender is an OutputStreamAppender that writes to the File named in the fileName parameter and rolls the file over according the TriggeringPolicy and the RolloverPolicy. The RollingFileAppender uses a RollingFileManager (which extends OutputStreamManager) to actually perform the file I/O and perform the rollover. While RolloverFileAppenders from different Configurations cannot be shared, the RollingFileManagers can be if the Manager is accessible. For example, two web applications in a servlet container can have their own configuration and safely write to the same file if Log4j is in a ClassLoader that is common to both of them.

A RollingFileAppender requires a TriggeringPolicy and a RolloverStrategy. The triggering policy determines if a rollover should be performed while the RolloverStrategy defines how the rollover should be done. If no RolloverStrategy is configured, RollingFileAppender will use the DefaultRolloverStrategy. Since log4j-2.5, a custom delete action can be configured in the DefaultRolloverStrategy to run at rollover. Since 2.8 if no file name is configured then DirectWriteRolloverStrategy will be used instead of DefaultRolloverStrategy. Since log4j-2.9, a custom POSIX file attribute view action can be configured in the DefaultRolloverStrategy to run at rollover, if not defined, inherited POSIX file attribute view from the RollingFileAppender will be applied.

File locking is not supported by the RollingFileAppender.

RollingFileAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| append | boolean | When true - the default, records will be appended to the end of the file. When set to false, the file will be cleared before new records are written. |
| bufferedIO | boolean | When true - the default, records will be written to a buffer and the data will be written to disk when the buffer is full or, if immediateFlush is set, when the record is written. File locking cannot be used with bufferedIO. Performance tests have shown that using buffered I/O significantly improves performance, even if immediateFlush is enabled. |

| | | |
|---|---|---|
| bufferSize | int | When bufferedIO is true, this is the buffer size, the default is 8192 bytes. |
| createOnDemand | boolean | The appender creates the file on-demand. The appender only creates the file when a log event passes all filters and is routed to this appender. Defaults to false. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| fileName | String | The name of the file to write to. If the file, or any of its parent directories, do not exist, they will be created. |
| filePattern | String | The pattern of the file name of the archived log file. The format of the pattern is dependent on the RolloverPolicy that is used. The DefaultRolloverPolicy will accept both a date/time pattern compatible with SimpleDateFormat (http://download.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html) and/or a %i which represents an integer counter. The pattern also supports interpolation at runtime so any of the Lookups (such as the DateLookup (./lookups.html#DateLookup)) can be included in the pattern. |
| immediateFlush | boolean | When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance.<br>Flushing after every write is only useful when using this appender with synchronous loggers. Asynchronous loggers and appenders will automatically flush at the end of a batch of events, even if immediateFlush is set to false. This also guarantees the data is written to disk but is more efficient. |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| name | String | The name of the Appender. |
| policy | TriggeringPolicy | The policy to use to determine if a rollover should occur. |
| strategy | RolloverStrategy | The strategy to use to determine the name and location of the archive file. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| filePermissions | String | File attribute permissions in POSIX format to apply whenever the file is created.<br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view.<br>Examples: rw------- or rw-rw-rw- etc... |
| fileOwner | String | File owner to define whenever the file is created.<br>Changing file's owner may be restricted for security reason and Operation not permitted IOException thrown. Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file if _POSIX_CHOWN_RESTRICTED (http://www.gnu.org/software/libc/manual/html_node/Options-for-Files.html) is in effect for path.<br>Underlying files system shall support file owner (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/FileOwnerAttributeView.html) attribute view. |
| fileGroup | String | File group to define whenever the file is created.<br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view. |

## Triggering Policies

#### Composite Triggering Policy

The CompositeTriggeringPolicy combines multiple triggering policies and returns true if any of the configured policies return true. The CompositeTriggeringPolicy is configured simply by wrapping other policies in a Policies element.

For example, the following XML fragment defines policies that rollover the log when the JVM starts, when the log size reaches twenty megabytes, and when the current date no longer matches the log's start date.

```
<Policies>
  <OnStartupTriggeringPolicy />
  <SizeBasedTriggeringPolicy size="20 MB" />
  <TimeBasedTriggeringPolicy />
</Policies>
```

#### Cron Triggering Policy

The CronTriggeringPolicy triggers rollover based on a cron expression. This policy is controlled by a timer and is asynchronous to processing log events, so it is possible that log events from the previous or next time period may appear at the beginning or end of the log file. The filePattern attribute of the Appender should contain a timestamp otherwise the target file will be overwritten on each rollover.

CronTriggeringPolicy Parameters

| Parameter Name | Type | Description |
|---|---|---|
| schedule | String | The cron expression. The expression is the same as what is allowed in the Quartz scheduler. See CronExpression (../log4j-core/apidocs/org/apache/logging/log4j/core/util/CronExpression.html) for a full description of the expression. |
| evaluateOnStartup | boolean | On startup the cron expression will be evaluated against the file's last modification timestamp. If the cron expression indicates a rollover should have occurred between that time and the current time the file will be immediately rolled over. |

#### OnStartup Triggering Policy

The OnStartupTriggeringPolicy policy causes a rollover if the log file is older than the current JVM's start time and the minimum file size is met or exceeded.

OnStartupTriggeringPolicy Parameters

| Parameter Name | Type | Description |
|---|---|---|
| minSize | long | The minimum size the file must have to roll over. A size of zero will cause a roll over no matter what the file size is. The default value is 1, which will prevent rolling over an empty file. |

*Google App Engine note:*
When running in Google App Engine, the OnStartup policy causes a rollover if the log file is older than *the time when Log4J initialized*. (Google App Engine restricts access to certain classes so Log4J cannot determine JVM start time with java.lang.management.ManagementFactory.getRuntimeMXBean().getStartTime() and falls back to Log4J initialization time instead.)

#### SizeBased Triggering Policy

The SizeBasedTriggeringPolicy causes a rollover once the file has reached the specified size. The size can be specified in bytes, with the suffix KB, MB or GB, for example 20MB. When combined with a time based triggering policy the file pattern must contain a %i otherwise the target file will be overwritten on every rollover as the SizeBased Triggering Policy will not cause the timestamp value in the file name to change. When used without a time based triggering policy the SizeBased Triggering Policy will cause the timestamp value to change.

#### TimeBased Triggering Policy

The TimeBasedTriggeringPolicy causes a rollover once the date/time pattern no longer applies to the active file. This policy accepts an interval attribute which indicates how frequently the rollover should occur based on the time pattern and a modulate boolean attribute.

TimeBasedTriggeringPolicy Parameters

| Parameter Name | Type | Description |
|---|---|---|
| interval | integer | How often a rollover should occur based on the most specific time unit in the date pattern. For example, with a date pattern with hours as the most specific item and and increment of 4 rollovers would occur every 4 hours. The default value is 1. |
| modulate | boolean | Indicates whether the interval should be adjusted to cause the next rollover to occur on the interval boundary. For example, if the item is hours, the current hour is 3 am and the interval is 4 then the first rollover will occur at 4 am and then next ones will occur at 8 am, noon, 4pm, etc. |
| maxRandomDelay | integer | Indicates the maximum number of seconds to randomly delay a rollover. By default, this is 0 which indicates no delay. This setting is useful on servers where multiple applications are configured to rollover log files at the same time and can spread the load of doing so across time. |

### Rollover Strategies

#### Default Rollover Strategy

The default rollover strategy accepts both a date/time pattern and an integer from the filePattern attribute specified on the RollingFileAppender itself. If the date/time pattern is present it will be replaced with the current date and time values. If the pattern contains an integer it will be incremented on each rollover. If the pattern contains both a date/time and integer in the pattern the integer will be incremented until the result of the date/time pattern changes. If the file pattern ends with ".gz", ".zip", ".bz2", ".deflate", ".pack200", or ".xz" the resulting archive will be compressed using the compression scheme that matches the suffix. The formats bzip2, Deflate, Pack200 and XZ require Apache Commons Compress (http://commons.apache.org/proper/commons-compress/) . In addition, XZ requires XZ for Java (http://tukaani.org/xz/java.html) . The pattern may also contain lookup references that can be resolved at runtime such as is shown in the example below.

The default rollover strategy supports three variations for incrementing the counter. To illustrate how it works, suppose that the min attribute is set to 1, the max attribute is set to 3, the file name is "foo.log", and the file name pattern is "foo-%i.log".

| Number of rollovers | Active output target | Archived log files | Description |
|---|---|---|---|
| 0 | foo.log | - | All logging is going to the initial file. |
| 1 | foo.log | foo-1.log | During the first rollover foo.log is renamed to foo-1.log. A new foo.log file is created and starts being written to. |
| 2 | foo.log | foo-2.log, foo-1.log | During the second rollover foo.log is renamed to foo-2.log. A new foo.log file is created and starts being written to. |
| 3 | foo.log | foo-3.log, foo-2.log, foo-1.log | During the third rollover foo.log is renamed to foo-3.log. A new foo.log file is created and starts being written to. |
| 4 | foo.log | foo-3.log, foo-2.log, foo-1.log | In the fourth and subsequent rollovers, foo-1.log is deleted, foo-2.log is renamed to foo-1.log, foo-3.log is renamed to foo-2.log and foo.log is renamed to foo-3.log. A new foo.log file is created and starts being written to. |

By way of contrast, when the fileIndex attribute is set to "min" but all the other settings are the same the "fixed window" strategy will be performed.

| Number of rollovers | Active output target | Archived log files | Description |
|---|---|---|---|
| 0 | foo.log | - | All logging is going to the initial file. |
| 1 | foo.log | foo-1.log | During the first rollover foo.log is renamed to foo-1.log. A new foo.log file is created and starts being written to. |
| 2 | foo.log | foo-1.log, foo-2.log | During the second rollover foo-1.log is renamed to foo-2.log and foo.log is renamed to foo-1.log. A new foo.log file is created and starts being written to. |

| | | | |
|---|---|---|---|
| 3 | foo.log | foo-1.log, foo-2.log, foo-3.log | During the third rollover foo-2.log is renamed to foo-3.log, foo-1.log is renamed to foo-2.log and foo.log is renamed to foo-1.log. A new foo.log file is created and starts being written to. |
| 4 | foo.log | foo-1.log, foo-2.log, foo-3.log | In the fourth and subsequent rollovers, foo-3.log is deleted, foo-2.log is renamed to foo-3.log, foo-1.log is renamed to foo-2.log and foo.log is renamed to foo-1.log. A new foo.log file is created and starts being written to. |

Finally, as of release 2.8, if the fileIndex attribute is set to "nomax" then the min and max values will be ignored and file numbering will increment by 1 and each rollover will have an incrementally higher value with no maximum number of files.

DefaultRolloverStrategy Parameters

| Parameter Name | Type | Description |
|---|---|---|
| fileIndex | String | If set to "max" (the default), files with a higher index will be newer than files with a smaller index. If set to "min", file renaming and the counter will follow the Fixed Window strategy described above. |
| min | integer | The minimum value of the counter. The default value is 1. |
| max | integer | The maximum value of the counter. Once this values is reached older archives will be deleted on subsequent rollovers. The default value is 7. |
| compressionLevel | integer | Sets the compression level, 0-9, where 0 = none, 1 = best speed, through 9 = best compression. Only implemented for ZIP files. |
| tempCompressedFilePattern | String | The pattern of the file name of the archived log file during compression. |

**DirectWrite Rollover Strategy**

The DirectWriteRolloverStrategy causes log events to be written directly to files represented by the file pattern. With this strategy file renames are not performed. If the size-based triggering policy causes multiple files to be written during the specified time period they will be numbered starting at one and continually incremented until a time-based rollover occurs.

Warning: If the file pattern has a suffix indicating compression should take place the current file will not be compressed when the application is shut down. Furthermore, if the time changes such that the file pattern no longer matches the current file it will not be compressed at startup either.

DirectWriteRolloverStrategy Parameters

| Parameter Name | Type | Description |
|---|---|---|
| maxFiles | String | The maximum number of files to allow in the time period matching the file pattern. If the number of files is exceeded the oldest file will be deleted. If specified, the value must be greater than 1. If the value is less than zero or omitted then the number of files will not be limited. |
| compressionLevel | integer | Sets the compression level, 0-9, where 0 = none, 1 = best speed, through 9 = best compression. Only implemented for ZIP files. |
| tempCompressedFilePattern | String | The pattern of the file name of the archived log file during compression. |

Below is a sample configuration that uses a RollingFileAppender with both the time and size based triggering policies, will create up to 7 archives on the same day (1-7) that are stored in a directory based on the current year and month, and will compress each archive using gzip:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingFile name="RollingFile" fileName="logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

This second example shows a rollover strategy that will keep up to 20 files before removing them.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingFile name="RollingFile" fileName="logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="20"/>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

Below is a sample configuration that uses a RollingFileAppender with both the time and size based triggering policies, will create up to 7 archives on the same day (1-7) that are stored in a directory based on the current year and month, and will compress each archive using gzip and will roll every 6 hours when the hour is divisible by 6:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingFile name="RollingFile" fileName="logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{yyyy-MM-dd-HH}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy interval="6" modulate="true"/>
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

This sample configuration uses a RollingFileAppender with both the cron and size based triggering policies, and writes directly to an unlimited number of archive files. The cron trigger causes a rollover every hour while the file size is limited to 250MB:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingFile name="RollingFile" filePattern="logs/app-%d{yyyy-MM-dd-HH}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <CronTriggeringPolicy schedule="0 0 * * * ?"/>
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

This sample configuration is the same as the previous but limits the number of files saved each hour to 10:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingFile name="RollingFile" filePattern="logs/app-%d{yyyy-MM-dd-HH}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <CronTriggeringPolicy schedule="0 0 * * * ?"/>
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
      <DirectWriteRolloverStrategy maxFiles="10"/>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

**Log Archive Retention Policy: Delete on Rollover**

Log4j-2.5 introduces a Delete action that gives users more control over what files are deleted at rollover time than what was possible with the DefaultRolloverStrategy max attribute. The Delete action lets users configure one or more conditions that select the files to delete relative to a base directory.

Note that it is possible to delete any file, not just rolled over log files, so use this action with care! With the testMode parameter you can test your configuration without accidentally deleting the wrong files.

Delete Parameters

| Parameter Name | Type | Description |
|---|---|---|
| basePath | String | *Required.* Base path from where to start scanning for files to delete. |
| maxDepth | int | The maximum number of levels of directories to visit. A value of 0 means that only the starting file (the base path itself) is visited, unless denied by the security manager. A value of Integer.MAX_VALUE indicates that all levels should be visited. The default is 1, meaning only the files in the specified base directory. |
| followLinks | boolean | Whether to follow symbolic links. Default is false. |
| testMode | boolean | If true, files are not deleted but instead a message is printed to the status logger (configuration.html#StatusMessages) at INFO level. Use this to do a dry run to test if the configuration works as expected. Default is false. |
| pathSorter | PathSorter | A plugin implementing the PathSorter (../log4j-core/apidocs/org/apache/logging/log4j/core/appender/rolling/action/PathSorter.html) interface to sort the files before selecting the files to delete. The default is to sort most recently modified files first. |
| pathConditions | PathCondition[] | *Required if no ScriptCondition is specified.* One or more PathCondition elements.<br><br>If more than one condition is specified, they all need to accept a path before it is deleted. Conditions can be nested, in which case the inner condition(s) are evaluated only if the outer condition accepts the path. If conditions are not nested they may be evaluated in any order.<br><br>Conditions can also be combined with the logical operators AND, OR and NOT by using the IfAll, IfAny and IfNot composite conditions.<br><br>Users can create custom conditions or use the built-in conditions:<br><br>• IfFileName - accepts files whose path (relative to the base path) matches a regular expression (https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html) or a glob (https://docs.oracle.com/javase/7/docs/api/java/nio/file/FileSystem.html#getPathMatcher(java.lang.String)) .<br>• IfLastModified - accepts files that are as old as or older than the specified duration (../log4j-core/apidocs/org/apache/logging/log4j/core/appender/rolling/action/Duration.html#parseCharSequence).<br>• IfAccumulatedFileCount - accepts paths after some count threshold is exceeded during the file tree walk.<br>• IfAccumulatedFileSize - accepts paths after the accumulated file size threshold is exceeded during the file tree walk.<br>• IfAll - accepts a path if all nested conditions accept it (logical AND). Nested conditions may be evaluated in any order.<br>• IfAny - accepts a path if one of the nested conditions accept it (logical OR). Nested conditions may be evaluated in any order.<br>• IfNot - accepts a path if the nested condition does not accept it (logical NOT). |
| scriptCondition | ScriptCondition | *Required if no PathConditions are specified.* A ScriptCondition element specifying a script.<br><br>The ScriptCondition should contain a Script, ScriptRef or ScriptFile element that specifies the logic to be executed. (See also the ScriptFilter (filters.html#Script) documentation for more examples of configuring ScriptFiles and ScriptRefs.)<br><br>The script is passed a number of parameters, including a list of paths found under the base path (up to maxDepth) and must return a list with the paths to delete. |

IfFileName Condition Parameters

| Parameter Name | Type | Description |
|---|---|---|
| glob | String | *Required if regex not specified.* Matches the relative path (relative to the base path) using a limited pattern language that resembles regular expressions but with a simpler syntax (https://docs.oracle.com/javase/7/docs/api/java/nio/file/FileSystem.html#getPathMatcher(java.lang.String)) . |

| | | |
|---|---|---|
| regex | String | *Required if glob not specified.* Matches the relative path (relative to the base path) using a regular expression as defined by the Pattern (https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html) class. |
| nestedConditions | PathCondition[] | An optional set of nested PathConditions. If any nested conditions exist they all need to accept the file before it is deleted. Nested conditions are only evaluated if the outer condition accepts a file (if the path name matches). |

IfLastModified Condition Parameters

| Parameter Name | Type | Description |
|---|---|---|
| age | String | *Required.* Specifies a duration (../log4j-core/apidocs/org/apache/logging/log4j/core/appender/rolling/action/Duration.html#parseCharSequence). The condition accepts files that are as old or older than the specified duration. |
| nestedConditions | PathCondition[] | An optional set of nested PathConditions. If any nested conditions exist they all need to accept the file before it is deleted. Nested conditions are only evaluated if the outer condition accepts a file (if the file is old enough). |

IfAccumulatedFileCount Condition Parameters

| Parameter Name | Type | Description |
|---|---|---|
| exceeds | int | *Required.* The threshold count from which files will be deleted. |
| nestedConditions | PathCondition[] | An optional set of nested PathConditions. If any nested conditions exist they all need to accept the file before it is deleted. Nested conditions are only evaluated if the outer condition accepts a file (if the threshold count has been exceeded). |

IfAccumulatedFileSize Condition Parameters

| Parameter Name | Type | Description |
|---|---|---|
| exceeds | String | *Required.* The threshold accumulated file size from which files will be deleted. The size can be specified in bytes, with the suffix KB, MB or GB, for example 20MB. |
| nestedConditions | PathCondition[] | An optional set of nested PathConditions. If any nested conditions exist they all need to accept the file before it is deleted. Nested conditions are only evaluated if the outer condition accepts a file (if the threshold accumulated file size has been exceeded). |

Below is a sample configuration that uses a RollingFileAppender with the cron triggering policy configured to trigger every day at midnight. Archives are stored in a directory based on the current year and month. All files under the base directory that match the "*/app-*.log.gz" glob and are 60 days old or older are deleted at rollover time.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Properties>
    <Property name="baseDir">logs</Property>
  </Properties>
  <Appenders>
    <RollingFile name="RollingFile" fileName="${baseDir}/app.log"
          filePattern="${baseDir}/$${date:yyyy-MM}/app-%d{yyyy-MM-dd}.log.gz">
      <PatternLayout pattern="%d %p %c{1.} [%t] %m%n" />
      <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
      <DefaultRolloverStrategy>
        <Delete basePath="${baseDir}" maxDepth="2">
          <IfFileName glob="*/app-*.log.gz" />
          <IfLastModified age="60d" />
        </Delete>
      </DefaultRolloverStrategy>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

Below is a sample configuration that uses a RollingFileAppender with both the time and size based triggering policies, will create up to 100 archives on the same day (1-100) that are stored in a directory based on the current year and month, and will compress each archive using gzip and will roll every hour. During every rollover, this configuration will delete files that match "*/app-*.log.gz" and are 30 days old or older, but keep the most recent 100 GB or the most recent 10 files, whichever comes first.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Properties>
    <Property name="baseDir">logs</Property>
  </Properties>
  <Appenders>
    <RollingFile name="RollingFile" fileName="${baseDir}/app.log"
          filePattern="${baseDir}/$${date:yyyy-MM}/app-%d{yyyy-MM-dd-HH}-%i.log.gz">
      <PatternLayout pattern="%d %p %c{1.} [%t] %m%n" />
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="100">
        <!--
        Nested conditions: the inner condition is only evaluated on files
        for which the outer conditions are true.
        -->
        <Delete basePath="${baseDir}" maxDepth="2">
          <IfFileName glob="*/app-*.log.gz">
            <IfLastModified age="30d">
              <IfAny>
                <IfAccumulatedFileSize exceeds="100 GB" />
                <IfAccumulatedFileCount exceeds="10" />
              </IfAny>
            </IfLastModified>
          </IfFileName>
        </Delete>
      </DefaultRolloverStrategy>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

ScriptCondition Parameters

| Parameter Name | Type | Description |
|---|---|---|
| script | Script, ScriptFile or ScriptRef | The Script element that specifies the logic to be executed. The script is passed a list of paths found under the base path and must return the paths to delete as a java.util.List<PathWithAttributes (../log4j-core/apidocs/org/apache/logging/log4j/core/appender/rolling/action/PathWithAttributes.html)>. See also the ScriptFilter (filters.html#Script) documentation for an example of how ScriptFiles and ScriptRefs can be configured. |

Script Parameters

| Parameter Name | Type | Description |
|---|---|---|
| basePath | java.nio.file.Path | The directory from where the Delete action started scanning for files to delete. Can be used to relativize the paths in the pathList. |
| pathList | java.util.List<PathWithAttributes (../log4j-core/apidocs/org/apache/logging/log4j/core/appender/rolling/action/PathWithAttributes.html)> | The list of paths found under the base path up to the specified max depth, sorted most recently modified files first. The script is free to modify and return this list. |
| statusLogger | StatusLogger | The StatusLogger that can be used to log internal events during script execution. |
| configuration | Configuration | The Configuration that owns this ScriptCondition. |
| substitutor | StrSubstitutor | The StrSubstitutor used to replace lookup variables. |
| ? | String | Any properties declared in the configuration. |

Below is a sample configuration that uses a RollingFileAppender with the cron triggering policy configured to trigger every day at midnight. Archives are stored in a directory based on the current year and month. The script returns a list of rolled over files under the base directory dated Friday the 13th. The Delete action will delete all files returned by the script.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="trace" name="MyApp" packages="">
  <Properties>
    <Property name="baseDir">logs</Property>
  </Properties>
  <Appenders>
    <RollingFile name="RollingFile" fileName="${baseDir}/app.log"
          filePattern="${baseDir}/$${date:yyyy-MM}/app-%d{yyyyMMdd}.log.gz">
      <PatternLayout pattern="%d %p %c{1.} [%t] %m%n" />
      <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
      <DefaultRolloverStrategy>
        <Delete basePath="${baseDir}" maxDepth="2">
          <ScriptCondition>
            <Script name="superstitious" language="groovy"><![CDATA[
                import java.nio.file.*;

                def result = [];
                def pattern = ~/\d*\/app-(\d*)\.log\.gz/;

                pathList.each { pathWithAttributes ->
                  def relative = basePath.relativize pathWithAttributes.path
                  statusLogger.trace 'SCRIPT: relative path=' + relative + " (base=$basePath)";

                  // remove files dated Friday the 13th

                  def matcher = pattern.matcher(relative.toString());
                  if (matcher.find()) {
                    def dateString = matcher.group(1);
                    def calendar = Date.parse("yyyyMMdd", dateString).toCalendar();
                    def friday13th = calendar.get(Calendar.DAY_OF_MONTH) == 13 \
                                  && calendar.get(Calendar.DAY_OF_WEEK) == Calendar.FRIDAY;
                    if (friday13th) {
                      result.add pathWithAttributes;
                      statusLogger.trace 'SCRIPT: deleting path ' + pathWithAttributes;
                    }
                  }
                }
                statusLogger.trace 'SCRIPT: returning ' + result;
                result;
              ]] >
            </Script>
          </ScriptCondition>
        </Delete>
      </DefaultRolloverStrategy>
    </RollingFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>
</Configuration>
```

#### Log Archive File Attribute View Policy: Custom file attribute on Rollover

Log4j-2.9 introduces a PosixViewAttribute action that gives users more control over which file attribute permissions, owner and group should be applied. The PosixViewAttribute action lets users configure one or more conditions that select the eligible files relative to a base directory.

PosixViewAttribute Parameters

| Parameter Name | Type | Description |
|---|---|---|
| basePath | String | *Required.* Base path from where to start scanning for files to apply attributes. |
| maxDepth | int | The maximum number of levels of directories to visit. A value of 0 means that only the starting file (the base path itself) is visited, unless denied by the security manager. A value of Integer.MAX_VALUE indicates that all levels should be visited. The default is 1, meaning only the files in the specified base directory. |
| followLinks | boolean | Whether to follow symbolic links. Default is false. |
| pathConditions | PathCondition[] | see DeletePathCondition |
| filePermissions | String | File attribute permissions in POSIX format to apply when action is executed.<br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view.<br>Examples: rw------- or rw-rw-rw- etc... |

| | | |
|---|---|---|
| fileOwner | String | File owner to define when action is executed.<br><br>Changing file's owner may be restricted for security reason and Operation not permitted IOException thrown. Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file if _POSIX_CHOWN_RESTRICTED (http://www.gnu.org/software/libc/manual/html_node/Options-for-Files.html) is in effect for path.<br><br>Underlying files system shall support file owner (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/FileOwnerAttributeView.html) attribute view. |
| fileGroup | String | File group to define when action is executed.<br><br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view. |

Below is a sample configuration that uses a RollingFileAppender and defines different POSIX file attribute view for current and rolled log files.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="trace" name="MyApp" packages="">
  <Properties>
    <Property name="baseDir">logs</Property>
  </Properties>
  <Appenders>
    <RollingFile name="RollingFile" fileName="${baseDir}/app.log"
                         filePattern="${baseDir}/$${date:yyyy-MM}/app-%d{yyyyMMdd}.log.gz"
                 filePermissions="rw-------">
      <PatternLayout pattern="%d %p %c{1.} [%t] %m%n" />
      <CronTriggeringPolicy schedule="0 0 0 * * ?"/>
      <DefaultRolloverStrategy stopCustomActionsOnError="true">
        <PosixViewAttribute basePath="${baseDir}/$${date:yyyy-MM}" filePermissions="r--r--r--">
                <IfFileName glob="*.gz" />
        </PosixViewAttribute>
      </DefaultRolloverStrategy>
    </RollingFile>
  </Appenders>

  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingFile"/>
    </Root>
  </Loggers>

</Configuration>
```

# RollingRandomAccessFileAppender

The RollingRandomAccessFileAppender is similar to the standard RollingFileAppender except it is always buffered (this cannot be switched off) and internally it uses a ByteBuffer + RandomAccessFile instead of a BufferedOutputStream. We saw a 20-200% performance improvement compared to RollingFileAppender with "bufferedIO=true" in our measurements (../performance.html#whichAppender). The RollingRandomAccessFileAppender writes to the File named in the fileName parameter and rolls the file over according the TriggeringPolicy and the RolloverPolicy. Similar to the RollingFileAppender, RollingRandomAccessFileAppender uses a RollingRandomAccessFileManager to actually perform the file I/O and perform the rollover. While RollingRandomAccessFileAppender from different Configurations cannot be shared, the RollingRandomAccessFileManagers can be if the Manager is accessible. For example, two web applications in a servlet container can have their own configuration and safely write to the same file if Log4j is in a ClassLoader that is common to both of them.

A RollingRandomAccessFileAppender requires a TriggeringPolicy and a RolloverStrategy. The triggering policy determines if a rollover should be performed while the RolloverStrategy defines how the rollover should be done. If no RolloverStrategy is configured, RollingRandomAccessFileAppender will use the DefaultRolloverStrategy. Since log4j-2.5, a custom delete action can be configured in the DefaultRolloverStrategy to run at rollover.

File locking is not supported by the RollingRandomAccessFileAppender.

RollingRandomAccessFileAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| append | boolean | When true - the default, records will be appended to the end of the file. When set to false, the file will be cleared before new records are written. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| fileName | String | The name of the file to write to. If the file, or any of its parent directories, do not exist, they will be created. |
| filePattern | String | The pattern of the file name of the archived log file. The format of the pattern should is dependent on the RolloverStrategu that is used. The DefaultRolloverStrategy will accept both a date/time pattern compatible with SimpleDateFormat (http://download.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html) and/or a %i which represents an integer counter. The integer counter allows specifying a padding, like %3i for space-padding the counter to 3 digits or (usually more useful) %03i for zero-padding the counter to 3 digits. The pattern also supports interpolation at runtime so any of the Lookups (such as the DateLookup (./lookups.html#DateLookup) can be included in the pattern. |

| | | |
|---|---|---|
| immediateFlush | boolean | When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance.<br><br>Flushing after every write is only useful when using this appender with synchronous loggers. Asynchronous loggers and appenders will automatically flush at the end of a batch of events, even if immediateFlush is set to false. This also guarantees the data is written to disk but is more efficient. |
| bufferSize | int | The buffer size, defaults to 262,144 bytes (256 * 1024). |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| name | String | The name of the Appender. |
| policy | TriggeringPolicy | The policy to use to determine if a rollover should occur. |
| strategy | RolloverStrategy | The strategy to use to determine the name and location of the archive file. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| filePermissions | String | File attribute permissions in POSIX format to apply whenever the file is created.<br><br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view.<br><br>Examples: rw------- or rw-rw-rw- etc... |
| fileOwner | String | File owner to define whenever the file is created.<br><br>Changing file's owner may be restricted for security reason and Operation not permitted IOException thrown. Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file if _POSIX_CHOWN_RESTRICTED (http://www.gnu.org/software/libc/manual/html_node/Options-for-Files.html) is in effect for path.<br><br>Underlying files system shall support file owner (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/FileOwnerAttributeView.html) attribute view. |
| fileGroup | String | File group to define whenever the file is created.<br><br>Underlying files system shall support POSIX (https://docs.oracle.com/javase/7/docs/api/java/nio/file/attribute/PosixFileAttributeView.html) file attribute view. |

### Triggering Policies

See RollingFileAppender Triggering Policies.

### Rollover Strategies

See RollingFileAppender Rollover Strategies.

Below is a sample configuration that uses a RollingRandomAccessFileAppender with both the time and size based triggering policies, will create up to 7 archives on the same day (1-7) that are stored in a directory based on the current year and month, and will compress each archive using gzip:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingRandomAccessFile name="RollingRandomAccessFile" fileName="logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
    </RollingRandomAccessFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingRandomAccessFile"/>
    </Root>
  </Loggers>
</Configuration>
```

This second example shows a rollover strategy that will keep up to 20 files before removing them.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingRandomAccessFile name="RollingRandomAccessFile" fileName="logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{MM-dd-yyyy}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
      <DefaultRolloverStrategy max="20"/>
    </RollingRandomAccessFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingRandomAccessFile"/>
    </Root>
  </Loggers>
</Configuration>
```

Below is a sample configuration that uses a RollingRandomAccessFileAppender with both the time and size based triggering policies, will create up to 7 archives on the same day (1-7) that are stored in a directory based on the current year and month, and will compress each archive using gzip and will roll every 6 hours when the hour is divisible by 6:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <RollingRandomAccessFile name="RollingRandomAccessFile" fileName="logs/app.log"
                 filePattern="logs/$${date:yyyy-MM}/app-%d{yyyy-MM-dd-HH}-%i.log.gz">
      <PatternLayout>
        <Pattern>%d %p %c{1.} [%t] %m%n</Pattern>
      </PatternLayout>
      <Policies>
        <TimeBasedTriggeringPolicy interval="6" modulate="true"/>
        <SizeBasedTriggeringPolicy size="250 MB"/>
      </Policies>
    </RollingRandomAccessFile>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="RollingRandomAccessFile"/>
    </Root>
  </Loggers>
</Configuration>
```

# RoutingAppender

The RoutingAppender evaluates LogEvents and then routes them to a subordinate Appender. The target Appender may be an appender previously configured and may be referenced by its name or the Appender can be dynamically created as needed. The RoutingAppender should be configured after any Appenders it references to allow it to shut down properly.

You can also configure a RoutingAppender with scripts: you can run a script when the appender starts and when a route is chosen for an log event.

RoutingAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| Filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| name | String | The name of the Appender. |
| RewritePolicy | RewritePolicy | The RewritePolicy that will manipulate the LogEvent. |
| Routes | Routes | Contains one or more Route declarations to identify the criteria for choosing Appenders. |
| Script | Script | This Script runs when Log4j starts the RoutingAppender and returns a String Route key to determine the default Route.<br>This script is passed the following variables:<br>RoutingAppender Script Parameters<br>**Parameter Name** / **Type** / **Description**<br>configuration / Configuration / The active Configuration.<br>staticVariables / Map / A Map shared between all script invocations for this appender instance. This is the same map passed to the Routes Script. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |

In this example, the script causes the "ServiceWindows" route to be the default route on Windows and "ServiceOther" on all other operating systems. Note that the List Appender is one of our test appenders, any appender can be used, it is only used as a shorthand.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN" name="RoutingTest">
  <Appenders>
    <Routing name="Routing">
      <Script name="RoutingInit" language="JavaScript"><![CDATA[
        importPackage(java.lang);
        System.getProperty("os.name").search("Windows") > -1 ? "ServiceWindows" : "ServiceOther";]]>
      </Script>
      <Routes>
        <Route key="ServiceOther">
          <List name="List1" />
        </Route>
        <Route key="ServiceWindows">
          <List name="List2" />
        </Route>
      </Routes>
    </Routing>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Routing" />
    </Root>
  </Loggers>
</Configuration>
```

### Routes

The Routes element accepts a single attribute named "pattern". The pattern is evaluated against all the registered Lookups and the result is used to select a Route. Each Route may be configured with a key. If the key matches the result of evaluating the pattern then that Route will be selected. If no key is specified on a Route then that Route is the default. Only one Route can be configured as the default.

The Routes element may contain a Script child element. If specified, the Script is run for each log event and returns the String Route key to use.

You must specify either the pattern attribute or the Script element, but not both.

Each Route must reference an Appender. If the Route contains a ref attribute then the Route will reference an Appender that was defined in the configuration. If the Route contains an Appender definition then an Appender will be created within the context of the RoutingAppender and will be reused each time a matching Appender name is referenced through a Route.

This script is passed the following variables:

RoutingAppender Routes Script Parameters

| Parameter Name | Type | Description |
|---|---|---|
| configuration | Configuration | The active Configuration. |
| staticVariables | Map | A Map shared between all script invocations for this appender instance. This is the same map passed to the Routes Script. |
| logEvent | LogEvent | The log event. |

In this example, the script runs for each log event and picks a route based on the presence of a Marker named "AUDIT".

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN" name="RoutingTest">
  <Appenders>
    <Console name="STDOUT" target="SYSTEM_OUT" />
    <Flume name="AuditLogger" compress="true">
      <Agent host="192.168.10.101" port="8800"/>
      <Agent host="192.168.10.102" port="8800"/>
      <RFC5424Layout enterpriseNumber="18060" includeMDC="true" appName="MyApp"/>
    </Flume>
    <Routing name="Routing">
      <Routes>
        <Script name="RoutingInit" language="JavaScript"><![CDATA[
          if (logEvent.getMarker() != null && logEvent.getMarker().isInstanceOf("AUDIT")) {
                return "AUDIT";
            } else if (logEvent.getContextMap().containsKey("UserId")) {
                return logEvent.getContextMap().get("UserId");
            }
            return "STDOUT";]]>
        </Script>
        <Route>
          <RollingFile
              name="Rolling-${mdc:UserId}"
              fileName="${mdc:UserId}.log"
              filePattern="${mdc:UserId}.%i.log.gz">
            <PatternLayout>
              <pattern>%d %p %c{1.} [%t] %m%n</pattern>
            </PatternLayout>
            <SizeBasedTriggeringPolicy size="500" />
          </RollingFile>
        </Route>
        <Route ref="AuditLogger" key="AUDIT"/>
        <Route ref="STDOUT" key="STDOUT"/>
      </Routes>
      <IdlePurgePolicy timeToLive="15" timeUnit="minutes"/>
    </Routing>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Routing" />
    </Root>
  </Loggers>
</Configuration>
```

### Purge Policy

The RoutingAppender can be configured with a PurgePolicy whose purpose is to stop and remove dormant Appenders that have been dynamically created by the RoutingAppender. Log4j currently provides the IdlePurgePolicy as the only PurgePolicy available for cleaning up the Appenders. The IdlePurgePolicy accepts 2 attributes; timeToLive, which is the number of timeUnits the Appender should survive without having any events sent to it, and timeUnit, the String representation of java.util.concurrent.TimeUnit which is used with the timeToLive attribute.

Below is a sample configuration that uses a RoutingAppender to route all Audit events to a FlumeAppender and all other events will be routed to a RollingFileAppender that captures only the specific event type. Note that the AuditAppender was predefined while the RollingFileAppenders are created as needed.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Flume name="AuditLogger" compress="true">
      <Agent host="192.168.10.101" port="8800"/>
      <Agent host="192.168.10.102" port="8800"/>
      <RFC5424Layout enterpriseNumber="18060" includeMDC="true" appName="MyApp"/>
    </Flume>
    <Routing name="Routing">
      <Routes pattern="$${sd:type}">
        <Route>
          <RollingFile name="Rolling-${sd:type}" fileName="${sd:type}.log"
                       filePattern="${sd:type}.%i.log.gz">
            <PatternLayout>
              <pattern>%d %p %c{1.} [%t] %m%n</pattern>
            </PatternLayout>
            <SizeBasedTriggeringPolicy size="500" />
          </RollingFile>
        </Route>
        <Route ref="AuditLogger" key="Audit"/>
      </Routes>
      <IdlePurgePolicy timeToLive="15" timeUnit="minutes"/>
    </Routing>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Routing"/>
    </Root>
  </Loggers>
</Configuration>
```

## SMTPAppender

Sends an e-mail when a specific logging event occurs, typically on errors or fatal errors.

The number of logging events delivered in this e-mail depend on the value of **BufferSize** option. The SMTPAppender keeps only the last BufferSize logging events in its cyclic buffer. This keeps memory requirements at a reasonable level while still delivering useful application context. All events in the buffer are included in the email. The buffer will contain the most recent events of level TRACE to WARN preceding the event that triggered the email.

The default behavior is to trigger sending an email whenever an ERROR or higher severity event is logged and to format it as HTML. The circumstances on when the email is sent can be controlled by setting one or more filters on the Appender. As with other Appenders, the formatting can be controlled by specifying a Layout for the Appender.

SMTPAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | The name of the Appender. |
| from | String | The email address of the sender. |
| replyTo | String | The comma-separated list of reply-to email addresses. |
| to | String | The comma-separated list of recipient email addresses. |
| cc | String | The comma-separated list of CC email addresses. |
| bcc | String | The comma-separated list of BCC email addresses. |
| subject | String | The subject of the email message. |
| bufferSize | integer | The maximum number of log events to be buffered for inclusion in the message. Defaults to 512. |
| layout | Layout | The Layout to use to format the LogEvent. If no layout is supplied HTML layout (layouts.html#HTMLLayout) will be used. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| smtpDebug | boolean | When set to true enables session debugging on STDOUT. Defaults to false. |
| smtpHost | String | The SMTP hostname to send to. This parameter is required. |
| smtpPassword | String | The password required to authenticate against the SMTP server. |
| smtpPort | integer | The SMTP port to send to. |
| smtpProtocol | String | The SMTP transport protocol (such as "smtps", defaults to "smtp"). |
| smtpUsername | String | The username required to authenticate against the SMTP server. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| SSL | SslConfiguration | Contains the configuration for the KeyStore and TrustStore. See SSL. |

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <SMTP name="Mail" subject="Error Log" to="errors@logging.apache.org" from="test@logging.apache.org"
          smtpHost="localhost" smtpPort="25" bufferSize="50">
    </SMTP>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="Mail"/>
    </Root>
  </Loggers>
</Configuration>
```

## ScriptAppenderSelector

When the configuration is built, the ScriptAppenderSelector appender calls a Script to compute an appender name. Log4j then creates one of the appender named listed under AppenderSet using the name of the ScriptAppenderSelector. After configuration, Log4j ignores the ScriptAppenderSelector. Log4j only builds the one selected appender from the configuration tree, and ignores other AppenderSet child nodes.

In the following example, the script returns the name "List2". The appender name is recorded under the name of the ScriptAppenderSelector, not the name of the selected appender, in this example, "SelectIt".

```
<Configuration status="WARN" name="ScriptAppenderSelectorExample">
  <Appenders>
    <ScriptAppenderSelector name="SelectIt">
      <Script language="JavaScript"><![CDATA[
        importPackage(java.lang);
        System.getProperty("os.name").search("Windows") > -1 ? "MyCustomWindowsAppender" : "MySyslogAppender";]]>
      </Script>
      <AppenderSet>
        <MyCustomWindowsAppender name="MyAppender" ... />
        <SyslogAppender name="MySyslog" ... />
      </AppenderSet>
    </ScriptAppenderSelector>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="SelectIt" />
    </Root>
  </Loggers>
</Configuration>
```

## SocketAppender

The SocketAppender is an OutputStreamAppender that writes its output to a remote destination specified by a host and port. The data can be sent over either TCP or UDP and can be sent in any format. You can optionally secure communication with SSL. Note that the TCP and SSL variants write to the socket as a stream and do not expect response from the target destination. Due to limitations in the TCP protocol that means that when the target server closes its connection some log events may continue to appear to succeed until a closed connection exception is raised, causing those events to be lost. If guaranteed delivery is required a protocol that requires acknowledgements must be used.

SocketAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | The name of the Appender. |
| host | String | The name or address of the system that is listening for log events. This parameter is required. If the host name resolves to multiple IP addresses the TCP and SSL variations will fail over to the next IP address when a connection is lost. |
| port | integer | The port on the host that is listening for log events. This parameter must be specified. |
| protocol | String | "TCP" (default), "SSL" or "UDP". |
| SSL | SslConfiguration | Contains the configuration for the KeyStore and TrustStore. See SSL. |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| immediateFail | boolean | When set to true, log events will not wait to try to reconnect and will fail immediately if the socket is not available. |
| immediateFlush | boolean | When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance. |
| bufferedIO | boolean | When true - the default, events are written to a buffer and the data will be written to the socket when the buffer is full or, if immediateFlush is set, when the record is written. |
| bufferSize | int | When bufferedIO is true, this is the buffer size, the default is 8192 bytes. |
| layout | Layout | The Layout to use to format the LogEvent. Required, there is no default. *New since 2.9, in previous versions SerializedLayout was default.* |

| | | |
|---|---|---|
| reconnectionDelayMillis | integer | If set to a value greater than 0, after an error the SocketManager will attempt to reconnect to the server after waiting the specified number of milliseconds. If the reconnect fails then an exception will be thrown (which can be caught by the application if ignoreExceptions is set to false). |
| connectTimeoutMillis | integer | The connect timeout in milliseconds. The default is 0 (infinite timeout, like Socket.connect() methods). |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |

This is an unsecured TCP configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Socket name="socket" host="localhost" port="9500">
      <JsonLayout properties="true"/>
    </Socket>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="socket"/>
    </Root>
  </Loggers>
</Configuration>
```

This is a secured SSL configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Socket name="socket" host="localhost" port="9500">
      <JsonLayout properties="true"/>
      <SSL>
        <KeyStore   location="log4j2-keystore.jks" passwordEnvironmentVariable="KEYSTORE_PASSWORD"/>
        <TrustStore location="truststore.jks"      passwordFile="${sys:user.home}/truststore.pwd"/>
      </SSL>
    </Socket>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="socket"/>
    </Root>
  </Loggers>
</Configuration>
```

# SSL Configuration

Several appenders can be configured to use either a plain network connection or a Secure Socket Layer (SSL) connection. This section documents the parameters available for SSL configuration.

SSL Configuration Parameters

| Parameter Name | Type | Description |
|---|---|---|
| protocol | String | SSL if omitted. See also Standard names (http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#SSLContext) . |
| KeyStore | KeyStore | Contains your private keys and certificates, and determines which authentication credentials to send to the remote host. |
| TrustStore | TrustStore | Contains the CA certificates of the remote counterparty. Determines whether the remote authentication credentials (and thus the connection) should be trusted. |
| verifyHostName | boolean | Toggles whether or not to perform host name verification. Defaults to false. |

### KeyStore

The keystore is meant to contain your private keys and certificates, and determines which authentication credentials to send to the remote host.

KeyStore Configuration Parameters

| Parameter Name | Type | Description |
|---|---|---|
| location | String | Path to the keystore file. |
| password | char[] | Plain text password to access the keystore. Cannot be combined with either passwordEnvironmentVariable or passwordFile. |
| passwordEnvironmentVariable | String | Name of an environment variable that holds the password. Cannot be combined with either password or passwordFile. |
| passwordFile | String | Path to a file that holds the password. Cannot be combined with either password or passwordEnvironmentVariable. |

| | | |
|---|---|---|
| type | String | Optional KeyStore type, e.g. JKS, PKCS12, PKCS11, BKS, Windows-MY/Windows-ROOT, KeychainStore, etc. The default is JKS. See also Standard types (http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#KeyStore) . |
| keyManagerFactoryAlgorithm | String | Optional KeyManagerFactory algorithm. The default is SunX509. See also Standard algorithms (http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#KeyManagerFactory) . |

### TrustStore

The trust store is meant to contain the CA certificates you are willing to trust when a remote party presents its certificate. Determines whether the remote authentication credentials (and thus the connection) should be trusted.

In some cases, they can be one and the same store, although it is often better practice to use distinct stores (especially when they are file-based).

TrustStore Configuration Parameters

| Parameter Name | Type | Description |
|---|---|---|
| location | String | Path to the keystore file. |
| password | char[] | Plain text password to access the keystore. Cannot be combined with either passwordEnvironmentVariable or passwordFile. |
| passwordEnvironmentVariable | String | Name of an environment variable that holds the password. Cannot be combined with either password or passwordFile. |
| passwordFile | String | Path to a file that holds the password. Cannot be combined with either password or passwordEnvironmentVariable. |
| type | String | Optional KeyStore type, e.g. JKS, PKCS12, PKCS11, BKS, Windows-MY/Windows-ROOT, KeychainStore, etc. The default is JKS. See also Standard types (http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#KeyStore) . |
| trustManagerFactoryAlgorithm | String | Optional TrustManagerFactory algorithm. The default is SunX509. See also Standard algorithms (http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#TrustManagerFactory) . |

### Example

```
...
    <SSL>
      <KeyStore   location="log4j2-keystore.jks" passwordEnvironmentVariable="KEYSTORE_PASSWORD"/>
      <TrustStore location="truststore.jks"      passwordFile="${sys:user.home}/truststore.pwd"/>
    </SSL>
...
```

## SyslogAppender

The SyslogAppender is a SocketAppender that writes its output to a remote destination specified by a host and port in a format that conforms with either the BSD Syslog format or the RFC 5424 format. The data can be sent over either TCP or UDP.

SyslogAppender Parameters

| Parameter Name | Type | Description |
|---|---|---|
| advertise | boolean | Indicates whether the appender should be advertised. |
| appName | String | The value to use as the APP-NAME in the RFC 5424 syslog record. |
| charset | String | The character set to use when converting the syslog String to a byte array. The String must be a valid Charset (http://download.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html) . If not specified, the default system Charset will be used. |
| connectTimeoutMillis | integer | The connect timeout in milliseconds. The default is 0 (infinite timeout, like Socket.connect() methods). |
| enterpriseNumber | integer | The IANA enterprise number as described in RFC 5424 (http://tools.ietf.org/html/rfc5424#section-7.2.2) |
| filter | Filter | A Filter to determine if the event should be handled by this Appender. More than one Filter may be used by using a CompositeFilter. |
| facility | String | The facility is used to try to classify the message. The facility option must be set to one of "KERN", "USER", "MAIL", "DAEMON", "AUTH", "SYSLOG", "LPR", "NEWS", "UUCP", "CRON", "AUTHPRIV", "FTP", "NTP", "AUDIT", "ALERT", "CLOCK", "LOCAL0", "LOCAL1", "LOCAL2", "LOCAL3", "LOCAL4", "LOCAL5", "LOCAL6", or "LOCAL7". These values may be specified as upper or lower case characters. |
| format | String | If set to "RFC5424" the data will be formatted in accordance with RFC 5424. Otherwise, it will be formatted as a BSD Syslog record. Note that although BSD Syslog records are required to be 1024 bytes or shorter the SyslogLayout does not truncate them. The RFC5424Layout also does not truncate records since the receiver must accept records of up to 2048 bytes and may accept records that are longer. |
| host | String | The name or address of the system that is listening for log events. This parameter is required. |
| id | String | The default structured data id to use when formatting according to RFC 5424. If the LogEvent contains a StructuredDataMessage the id from the Message will be used instead of this value. |
| ignoreExceptions | boolean | The default is true, causing exceptions encountered while appending events to be internally logged and then ignored. When set to false exceptions will be propagated to the caller, instead. You must set this to false when wrapping this Appender in a FailoverAppender. |
| immediateFail | boolean | When set to true, log events will not wait to try to reconnect and will fail immediately if the socket is not available. |

| | | |
|---|---|---|
| immediateFlush | boolean | When set to true - the default, each write will be followed by a flush. This will guarantee the data is written to disk but could impact performance. |
| includeMDC | boolean | Indicates whether data from the ThreadContextMap will be included in the RFC 5424 Syslog record. Defaults to true. |
| Layout | Layout | A custom layout which overrides the format setting. |
| loggerFields | List of KeyValuePairs | Allows arbitrary PatternLayout patterns to be included as specified ThreadContext fields; no default specified. To use, include a >LoggerFields< nested element, containing one or more >KeyValuePair< elements. Each >KeyValuePair< must have a key attribute, which specifies the key name which will be used to identify the field within the MDC Structured Data element, and a value attribute, which specifies the PatternLayout pattern to use as the value. |
| mdcExcludes | String | A comma separated list of mdc keys that should be excluded from the LogEvent. This is mutually exclusive with the mdcIncludes attribute. This attribute only applies to RFC 5424 syslog records. |
| mdcIncludes | String | A comma separated list of mdc keys that should be included in the FlumeEvent. Any keys in the MDC not found in the list will be excluded. This option is mutually exclusive with the mdcExcludes attribute. This attribute only applies to RFC 5424 syslog records. |
| mdcRequired | String | A comma separated list of mdc keys that must be present in the MDC. If a key is not present a LoggingException will be thrown. This attribute only applies to RFC 5424 syslog records. |
| mdcPrefix | String | A string that should be prepended to each MDC key in order to distinguish it from event attributes. The default string is "mdc:". This attribute only applies to RFC 5424 syslog records. |
| messageId | String | The default value to be used in the MSGID field of RFC 5424 syslog records. |
| name | String | The name of the Appender. |
| newLine | boolean | If true, a newline will be appended to the end of the syslog record. The default is false. |
| port | integer | The port on the host that is listening for log events. This parameter must be specified. |
| protocol | String | "TCP" or "UDP". This parameter is required. |
| SSL | SslConfiguration | Contains the configuration for the KeyStore and TrustStore. See SSL. |
| reconnectionDelayMillis | integer | If set to a value greater than 0, after an error the SocketManager will attempt to reconnect to the server after waiting the specified number of milliseconds. If the reconnect fails then an exception will be thrown (which can be caught by the application if ignoreExceptions is set to false). |

A sample syslogAppender configuration that is configured with two SyslogAppenders, one using the BSD format and one using RFC 5424.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <Syslog name="bsd" host="localhost" port="514" protocol="TCP"/>
    <Syslog name="RFC5424" format="RFC5424" host="localhost" port="8514"
            protocol="TCP" appName="MyApp" includeMDC="true"
            facility="LOCAL0" enterpriseNumber="18060" newLine="true"
            messageId="Audit" id="App"/>
  </Appenders>
  <Loggers>
    <Logger name="com.mycorp" level="error">
      <AppenderRef ref="RFC5424"/>
    </Logger>
    <Root level="error">
      <AppenderRef ref="bsd"/>
    </Root>
  </Loggers>
</Configuration>
```

For SSL this appender writes its output to a remote destination specified by a host and port over SSL in a format that conforms with either the BSD Syslog format or the RFC 5424 format.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="warn" name="MyApp" packages="">
  <Appenders>
    <TLSSyslog name="bsd" host="localhost" port="6514">
      <SSL>
        <KeyStore   location="log4j2-keystore.jks" passwordEnvironmentVariable="KEYSTORE_PASSWORD"/>
        <TrustStore location="truststore.jks"      passwordFile="${sys:user.home}/truststore.pwd"/>
      </SSL>
    </TLSSyslog>
  </Appenders>
  <Loggers>
    <Root level="error">
      <AppenderRef ref="bsd"/>
    </Root>
  </Loggers>
</Configuration>
```

# ZeroMQ/JeroMQ Appender

The ZeroMQ appender uses the JeroMQ (https://github.com/zeromq/jeromq) library to send log events to one or more ZeroMQ endpoints.

This is a simple JeroMQ configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration name="JeroMQAppenderTest" status="TRACE">
  <Appenders>
    <JeroMQ name="JeroMQAppender">
      <Property name="endpoint">tcp://*:5556</Property>
      <Property name="endpoint">ipc://info-topic</Property>
    </JeroMQ>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="JeroMQAppender"/>
    </Root>
  </Loggers>
</Configuration>
```

The table below describes all options. Please consult the JeroMQ and ZeroMQ documentation for details.

JeroMQ Parameters

| Parameter Name | Type | Description |
|---|---|---|
| name | String | The name of the Appender. Required. |
| Layout | layout | The Layout to use to format the LogEvent. If no layout is supplied the default pattern layout of "%m%n" will be used. |
| Filters | Filter | The Filter(s) of the Appender. |
| Properties | Property[] | One or more Property elements, named endpoint. |
| ignoreExceptions | boolean | If true, exceptions will be logged and suppressed. If false errors will be logged and then passed to the application. |
| affinity | long | The ZMQ_AFFINITY option. Defaults to 0. |
| backlog | long | The ZMQ_BACKLOG option. Defaults to 100. |
| delayAttachOnConnect | boolean | The ZMQ_DELAY_ATTACH_ON_CONNECT option. Defaults to false. |
| identity | byte[] | The ZMQ_IDENTITY option. Defaults to none. |
| ipv4Only | boolean | The ZMQ_IPV4ONLY option. Defaults to true. |
| linger | long | The ZMQ_LINGER option. Defaults to -1. |
| maxMsgSize | long | The ZMQ_MAXMSGSIZE option. Defaults to -1. |
| rcvHwm | long | The ZMQ_RCVHWM option. Defaults to 1000. |
| receiveBufferSize | long | The ZMQ_RCVBUF option. Defaults to 0. |
| receiveTimeOut | int | The ZMQ_RCVTIMEO option. Defaults to -1. |
| reconnectIVL | long | The ZMQ_RECONNECT_IVL option. Defaults to 100. |
| reconnectIVLMax | long | The ZMQ_RECONNECT_IVL_MAX option. Defaults to 0. |
| sendBufferSize | long | The ZMQ_SNDBUF option. Defaults to 0. |
| sendTimeOut | int | The ZMQ_SNDTIMEO option. Defaults to -1. |
| sndHwm | long | The ZMQ_SNDHWM option. Defaults to 1000. |
| tcpKeepAlive | int | The ZMQ_TCP_KEEPALIVE option. Defaults to -1. |
| tcpKeepAliveCount | long | The ZMQ_TCP_KEEPALIVE_CNT option. Defaults to -1. |
| tcpKeepAliveIdle | long | The ZMQ_TCP_KEEPALIVE_IDLE option. Defaults to -1. |
| tcpKeepAliveInterval | long | The ZMQ_TCP_KEEPALIVE_INTVL option. Defaults to -1. |
| xpubVerbose | boolean | The ZMQ_XPUB_VERBOSE option. Defaults to false. |