LogMon Manual

Thomas Weidlich
E-Mail: thomas.weidlich@die-moesch.de

August 14, 2012

Contents

1 Introduction

1.1 Description

LogMon is a simple logfile monitor written in *Java* . The main features are:

- Use regex
- Modular escalation
- Local correlation in *JavaScript*
- Configuration in single xml file
- No installation
- Store last reading position

The core logfile engine watches all logfiles from configuration (XML) and alert by configurated modul to receiver. At moment two modules available: EMail or Tivoli.

LogMon allow the usage of *JavaScript* to control a escalation. Add a `<condition>` tag in each `<pattern>` tag inside configuration to define the script name. The Logfile engine will run the *JavaScript* every time where the pattern match. The Returncode of *JavaScript* enable or disable the escalation of current matched logfile line. There is a persistant memory database inside the core to decide from *JavaScript* between two or more matched lines of all watched logfiles. The feature make **LogMon** differend to simple "tail and grep" logfile monitoring. The database enable a local state correlation.

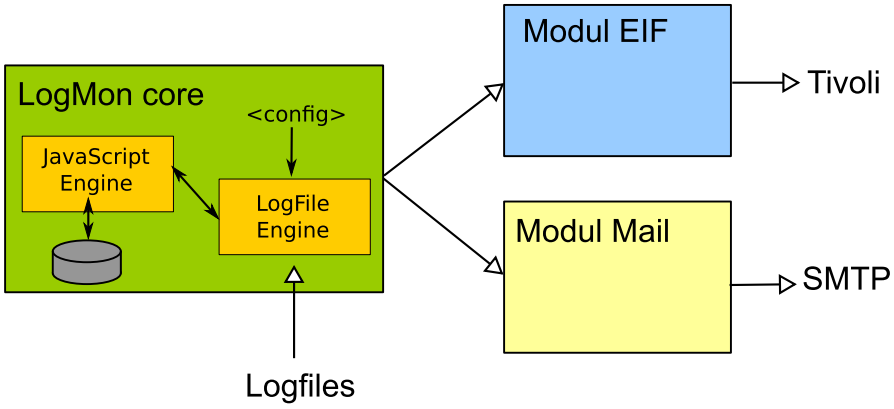

Figure 1: The LogMon Blockdiagram

1.2 Licence

This software is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation version 3 of the License.

This package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

2 Build and Installation

2.1 Build

To build **LogMon** core and module from source you need a jdk6 or newer and ant. Simply extract the source to some folder an call ant from commandline. The jar created inside folder dist.

2.2 Install

Simply copy logmon.jar and one alert module jar in a folder. Create a configuration file and start **LogMon** with *Java* version 6 or later (See example below). If the alert module require some additional jar, copy also into the folder.

```
java -cp logmonModMail.jar:logmon.jar:lib/mail.jar \
    app.LogMon --config=config.xml
```

3 The configuration

LogMon require one (validated by **LogMon**) xml file for configuration. Use argument `--config=filename.xml` to set filename. Inside of configuration there are three section inside of configuration file. See chapters blow for documentation.

- database
- alert
- logfiles

3.1 The section: database

The database section set name and path of internal database files. The database files store events for correlation from *JavaScript* and last reading positions. The path and pid entrys are optional. Examples:

```
<database>
    <id>LinuxBase</id>
    <path>/tmp</path>
    <pid>/tmp/logmon.pid</pid>
</database>
```

If `<path>` not given, the current folder is used. If `<pid>` is given, **LogMon** will check for file exists on start. If pid file already exists, **LogMon** will not start!

3.2 The section: alert

The way-of-alert is set in alert section. You have to add one alert module to *Java classpath* and setup the alerting in alert section. Today are two alert modules available:

Mail	logmonModmail.jar
Tivoli TEC	logmonModEIF.jar

See modules chapter at page 6 for additional alert configuration entry's. At this point a simple example

```
<alert>
    <class>mon.evt.StdoutAlert</class>
    <properties>
        <property name="init.msg"></property>
        <property name="send.pre">ERROR></property>
    </properties>
</alert>
```

To implement more alert modules you need *Java* Known-How and the alert interface from logmon source.

3.3 The section: logfile

The most important part of configuration are the one or more sections `<logfile>`. You can setup one or more logfiles with different pattern definitions! One first simple minimal example:

```
  <logfile>
    <file start="begin">test.log</file>
    <pattern>
      <regex>.*Error(.*)</regex>
      <msg>Error: $1</msg>
      <severity>CRITICAL</severity>
    </pattern>
  </logfile>
```

The first entry below `<logfile>` is the `<file>` entry to describe the logfile path. You can use environment variables in perl syntax `$ENV{HOME}/myfile.log`. The attribute start set the reading-start-position of logmon. You can use BEGIN or CURRENT or LAST. After the `<file>` section follow one more `<pattern>` definitions. Each `<pattern>` require the three child `<regex>`, `<msg>` an `<severity>`. Optional there are one `<condition>` and one `<properties>` child.

The `<regex>` contain a regular expression with or without groups. The matched group values available in `<msg>` as $1 for first group and $2 for second group and so on. The groups also available in conditions scripts. See description blow.

The resolved `<msg>` will send to alert receiver. You can use groups from regular expression and environment variables in perl syntax. The resolved message is changeable from condition script.

The valid severity's are INFO,WARNING,MINOR,CRITICAL.

The next example at page 5 a additional child `<condition>`. It set the filename of some *JavaScript* file. This file will call on every match of this pattern. If a condition script is given the alert is only send if the pattern matches **and** the condition script return `status=true`. See chapter 4

If no condition script is given the alert will send every time the pattern match.

The next optional child of `<logfile>` is `<properties>`. You can change every init and send property of **this** alert.

```
<logfile>
  <file start="current">/var/log/messages</file>
  <pattern>
    <regex>(\w+)\s+(\d+)\s+([\d:]+).*Deny Policy.*SRC=([\d\.]+).*DPT=(\d+)</regex>
    <msg>Firewall deny from IP $4 to Port $5</msg>
    <severity>WARNING</severity>
    <condition>firewall.js</condition>
  </pattern>
  <pattern>
    <regex>.*Error(.*)</regex>
    <msg>Error: $1</msg>
    <severity>CRITICAL</severity>
    <properties>
        <property name="send.smtp.subject">Error from Logfile</property>
    </properties>
  </pattern>
</logfile>
```

4 Condition scripts

The condition script allow a local correlation of logfile entry's before see alert to any receiver. You can send only every 5 entry for example or send only if some other entry was found in the past. Or you send only if more then 10 match in 2 hour. The condition script are written in *JavaScript* and run every matched line. the value of the variable *status* decide to send alert. If the *JavaScript* set *status=true*, the alert will send. You can use every *JavaScript* code inside. You have for,while,if and so one. There are some variables and instances already set by **LogMon** .

Name	Type	Description
pattern	String	The matched regular expression (Readonly)
logline	String	The current line from logfile. (Readonly)
status	String	The script return status. Default is false. If set to true the alert method will call
msg	String	The resolved messages. The script can change it.
db	Object	Database connection with method db.save(id) and db.load(id). Both method use/change/update the variable occurrence
occurrence	Object	The occurrence instance contains the attributes created,modified,maxage,repeat and groups. See Occurrence description

Table 1: Internal variables and instances

The simple thing first: The variable *pattern* and *logline* are read only and contain the `<regex>` from configuration and the current matched line from logfile. The *msg* contain the resolved message from configuration with all groups from `<regex>`. You can change it inside of *JavaScript* . The *status* is set to *false* and only if the *JavaScript* code change it to *true* the alert will send!

OK, lets got to next level. The next entry in table is the db instance with method *save(<id>)*, *load(<id>)* and *remove(<id>)*. This three method are the interface to a small database for store *occurrence* instance. A *occurrence* instance is created on every time a regex matched and you can save it inside the database. If a entry with same id already exists it will updated. The repeat will

increase, the modified time is changed and the groups[] is change to current match. If the current time is greater (later) then created+maxage (default 7 days) the entry is remove automatic, but you can remove every time. All condition scripts can access all database entry's by select the entry id's.

Attribute	Description
created	Epoch second of creation
modified	Epoch second of last modified
maxage	Maximal seconds store in database. The database entry will remove if `created+maxage > now`
repeat	The repeat count is increased while every db.save(id) action
groups[]	String array of all matched groups from regex. The index start with 0 (zero)!!

Table 2: The occurrence object

The used database is in memory but there is a mirror one file-system. The name and path are give in first section in configuration file. See chapter 3.1. It is a simple CSV file but you can't change it while **LogMon** is running!

Now let's look one first example. The follow example look for firewall line on a Linux system. The regex from configuration is `(\w+)\s+(\d+)\s+([\d:]+).*Deny Policy.*SRC=([\d\.]+).*DPT=(\d+)` and you see the fourth (index 3!!) regex-group contain the IP of the dropped connection. I use this IP as database index. I change the maxage to 1h check the repeat counter after saving in db.

The result is: Alert if there more as 5 connection from same IP in 1 hour then alert every 5 occurrence.

```
var src=occurrence.groups[3];
occurrence.maxage=3600;

db.save("FW"+src);

if(occurrence.repeat >=5  && occurrence.repeat % 5 ==0){
        status=true;
}
```

Note the % is the modulo operation in *JavaScript*

5 Modules

5.1 Setting of all modules

Every module implements a IAlert interface with three method init() send() stop(). All three method have a Properties instance as argument. The init() method all properties stating with init, the send() all properties starting with send and the stop() method starting with stop. The table below show all properties used by all module. You can set this properties (except send.repeat) inside section `<alert>` or inside section `<pattern>` of configuration file.

For module additional properties see chapters below.

Property	Description
send.repeat	The repeat count
send.severity	The severity
send.hostname	The hostname of problem location
send.msg	The message

For all *Java* programmer the interface. For more information see **LogMon** source.

```
package mon.evt;

import java.util.Properties;

public interface IAlert {

  public boolean init(Properties properties);

  public boolean send(Properties properties);

  public boolean stop();
}
```

5.2 Alert module: STDOUT

The stdout module alerts to the stdout console. I use it for debugging.

Property	Description
init.msg	Show this string at start of logmon
send.pre	Show this string at every line

```
<alert>
  <class>mon.evt.StdoutAlert</class>
  <properties>
    <property name="init.msg">Hello World</property>
    <property name="send.pre">Alert: </property>
  </properties>
</alert>
```

5.3 Alert module IBM Tivoli EIF

Send Event to IBM Tivoli Enterprise Console. It require some additional jar in folder lib. Contact IBM and ask for EIF SDK. The most important the properties init.tec1 and init.tec2. Setup two "TEC Gateway Reciever" (most time TEC servers).

```
<properties>
  <property name="init.tec1">tecgw1.tivoli.net</property>
  <property name="init.tec2">tecgw2.tivoli.net</property>
  <property name="send.class">MY_Event</property>
</properties>
```

5.4 Alert module Mail

Send alert as mail to SMTP receiver. Require the additional jar mail in CLASSPATH. If the locahost has listen at smtp port you only require init.smtp.server and I recommend init.smtp.from. :-)

Inside the pattern you will use send.smtp.to and send.smtp.subject as you like an need. The mail body is hardcoded. Maybe the future will give some template function.

If the SMTP server need some authentication set init.smtp.user and init.smtp.password. Be careful about file permissions of configuration! This is not a secure setup.

Property	Description
init.smtp.server	The smtp server
init.smtp.from	Sender mail address
init.smtp.user	The user if smtp need authentication
init.smtp.pwd	The password if smtp need authentication
send.smtp.to	Receiver mail address
send.smtp.subject	The Mail subject

Table 3: Modul mail properties

6 Debugging

6.1 logging

To write a logfile for debugging create logging.properties file an add **-Djava.util.logging.config.file = logging.properties** to arguments. For more information see *Java* logging documentation. Example properties file:

```
#
# Add -Djava.util.logging.config.file=logging.properties to vm arguments
#
handlers= java.util.logging.FileHandler,java.util.logging.ConsoleHandler

.level=INFO

#app.level=ALL
#cfg.level=ALL
#mon.level=ALL
#alert.level=ALL

# Setup handlers
java.util.logging.ConsoleHandler.level = INFO

java.util.logging.FileHandler.level=ALL
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
java.util.logging.FileHandler.pattern=logmon.log
java.util.logging.FileHandler.limit=102400
java.util.logging.FileHandler.count=5
java.util.logging.FileHandler.append=false
```

6.2 JMX (*Java* Management Extensions)

I add some JMX MBean to **LogMon** . To monitor **LogMon** from remote add follow arguments. It open some port without any security. Be careful! If you need some secure connection setting, please show *Java* jmx documentation.

```
-Djava.net.preferIPv4Stack=true
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9494
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

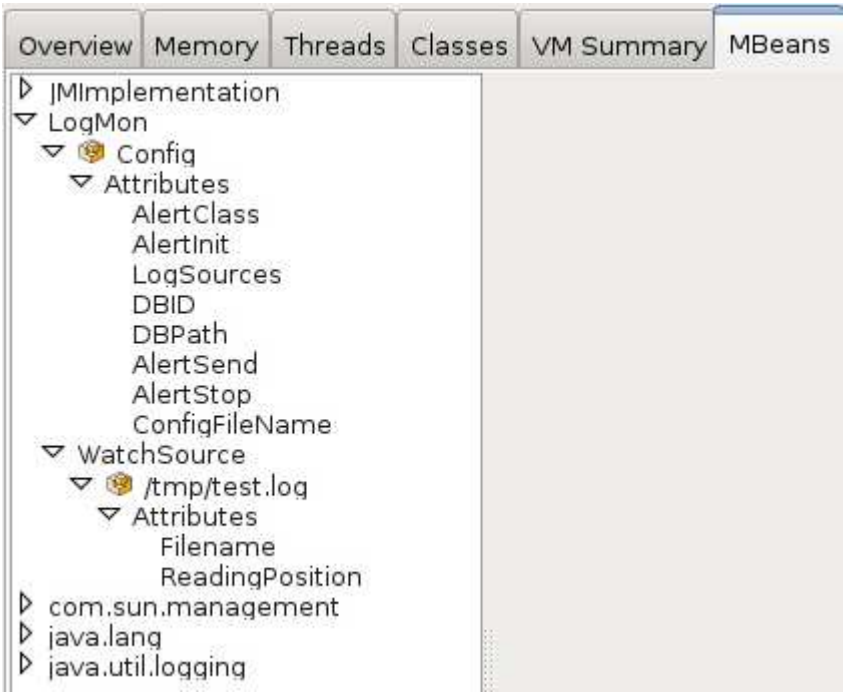

Figure 2: jconsole

You can use ssh to tunnel the JMX connecting from your workstation to **LogMon** server. First connect to the server by ssh. Add *-D <some-free-port>* to ssh.

```
ssh -D 9494 myhost.com
```

Now you start a jmx console at your workstation and connect it to the **LogMon** server. For *jconsole* use:

```
jconsole -J-DsocksProxyHost=localhost -J-DsocksProxyPort=9494 \
service:jmx:rmi:///jndi/rmi://myhost.com:9494/jmxrmi
```

Or or *jvisualvm* use:

```
jvisualvm \
    -J-Dnetbeans.system_socks_proxy=localhost:9494 \
    -J-Djava.net.useSystemProxies=true
```

And then:

- Add Remote Host (myhost.com)
- Add JMX Connection (myhost.com:9494)

A Configuration example files

Monitor Linux syslog an send alert to Tivoli TEC:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE config SYSTEM "configuration.dtd">
<config>
  <database>
    <id>LinuxTest</id>
  </database>
  <alert>
    <class>alert.Tivoli</class>
    <properties>
      <property name="init.tec1">gateway1.tivoli.net</property>
      <property name="init.tec2">gateway2.tivoli.net</property>
      <property name="send.class">MY_Class</property>
      <property name="send.hostname">MYHOST</property>
    </properties>
  </alert>
  <logfile>
    <!-- Start position at current or begin -->
    <file start="current">/var/log/messages</file>
    <pattern>
      <regex>(\w+)\s+(\d+)\s+([\d:]+).*Deny Policy.*SRC=([\d\.]+).*DPT=(\d+)</regex>
      <msg>Firewall deny from IP $4 to Port $5</msg>
      <severity>WARNING</severity>
      <condition>firewall.js</condition>
    </pattern>
    <pattern>
      <regex>.*Error(.*)</regex>
      <msg>Error: $1</msg>
      <severity>CRITICAL</severity>
      <properties>
        <property name="send.slot.srcip">$1</property>
        <property name="send.slot.appl">Firewall</property>
      </properties>
    </pattern>
  </logfile>
</config>
```

Monitor logfile and print alert to stdout:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "configuration.dtd">
<config>
  <database>
    <id>WinTest</id>
    <path>C:\temp</path>
  </database>
  <alert>
    <class>mon.evt.StdoutAlert</class>
    <properties>
      <property name="init.msg"></property>
      <property name="send.pre">ERROR></property>
    </properties>
  </alert>
  <logfile>
    <file start="last">test.log</file>
```

```
    <pattern>
      <regex>.*Error:(.*)</regex>
      <msg>Error: $1</msg>
      <severity>WARNING</severity>
      <condition>test.js</condition>
    </pattern>
  </logfile>
</config>
```