What is emond? - Magnusviri

By Magnusviri

Archived: 2026-04-06 01:49:47 UTC

I've been struggling with my log scanner that emails and sends SMS texts and I wondered if there was a better way. I found that emond does basically the same thing and found little documentation about it (but enough that I could figure out how to use it). So I got on one of those tangents and figured out so much that I felt compelled to write it down so that when I forget it all I can find better documentation if I ever wanted to use it again.

The name emond is short for Event Monitor Daemon. It is an OS X command located at /sbin/emond. I think it was added to OS X Server around 10.5 (2007). It was added to regular OS X in 10.7 (2011). There is very little documentation about this command (a short man page, a half a page in the 10.6 Server Admin guide, and a mention in a Peachpit book).

There are a few things that imply this tool has been neglected by Apple. First, it's has had bugs. Doing a websearch for emond brings up many webpages of people who have trouble with emond either taking 100% of the processor, either crashing (and restarting over and over thanks to launchd), or it fills up the system.log file with messages. I've seen this myself but because of the lack of documentation and my lack of understanding of what it was I fixed it with a restart.

Second, a supporting script at /usr/libexec/emlog.pl looks sloppy. There are sections that were commented out in 10.9 with mention that the functionality was replaced by the audit mechanism and it's still the same in 10.11. There are variables that are declared but not used and even a whole subroutine that isn't used (it was used in 10.8 and below). There is also the text " (for now...)" tacked onto an explanation, indicating that some functionality is likely to change in the future, a change that never occurred.

I have been able to figure a little about how emond works and it just has the same plist configuration driven feel of launchd. This is just a guess, but it looks to me like emond was meant to be a full featured notification system that was conceived around the same time as launchd and the Apple System Logger (ASL), both which were introduced in 10.4. I also think it was eventually suppose to be documented more so that others could use it (like me).

Apple changed directions between 10.6 and 10.7. I don't know what happened inside of the company (ok, it was the iPhone) and I haven't really thought all of the changes out, but 10.5 (released Oct 2007) and 10.6 (released Aug 2009) were amazingly well documented. And of course 10.7 (released Jul 2011) was when OS X Server became an application instead of an OS, lost a large portion of it's functionality, and there hasn't really been much documentation since 10.6. I think emond was affected by this change.

The whole time I've written this I've debated if this is worth documenting as much as I am especially because Apple should document it and also because 10.9 OS X Server almost migrated away from it completely. Maybe it will go away in a future OS. But I know I'll forget all of this once I go back to my regular tasks and it still works in 10.11 and I might want to use it more than just what I've already decided to do.

I've mostly looked at 10.8 Server, 10.9, 10.9 Server, and 10.11. I briefly looked at 10.7 to see if it was there (it was--probably because the server OS version was replaced by the app so they had to move this to the regular OS--and suddenly I'm wondering just how much of 10.6 Server was moved to 10.7...). It's not in 10.6 (non-server version). I don't have easy access to the server versions of 10.5, 10.6, 10.10 or 10.11 right now, but based on a comment on a webpage I think it first showed up in 10.5 Server.

So, here is what I've found, presented as quickly as possible.

What starts emond (launchd)

It all starts with launchd. The plist at /System/Library/LaunchDaemons/com.apple.emond.plist specifies the QueueDirectories of /private/var/db/emondClients. On my 10.9 OS X Server, there is an empty file in there named com.apple.server. So if something is in that directory, emond starts up. I just ran this command on a non-server to start it.

```
sudo touch /private/var/db/emondClients/bla
```

emond's config and rule files

When emond starts up, it reads it's plist file at /etc/emond.d/emond.plist. There is a man page for this file, which suggest to me that Apple intended on others to modify it (there are many missing man pages, I can't see why Apple would make one unless it was done by a developer on his own initiative). This plist file doesn't really contain anything remarkable. On my 10.9 OS X Server the following line is added, which tells it to look for more rules in the Server.app bundle.

```
 <key>additionalRulesPaths</key>
<array>
    <string>/Applications/Server.app/Contents/ServerRoot/private/etc/emond.d/rules/</string>
```

After it reads it's pref file it reads all of the files located in /etc/emond.d/rules/. All of my computers have this file in it, which is disabled by default.

- SampleRules.plist

Some of my computers have this file. I didn't see a pattern to explain why some have it and others don't.

- Xsan.plist

On my 10.8 Server I have these files in /Applications/Server.app/Contents/ServerRoot/private/etc/emond.d/rules/.

- AdaptiveFirewall.plist
- DHABlock.plist
- DiskStatus.plist
- EmondCertificateExpiring.plist
- HostBlockingLogic.plist
- NetworkAlertControl.plist
- com.apple.assetcache.plist
- com.apple.disks.disappeared.plist
- com.apple.disks.smart.status.plist
- com.apple.disks.space.plist
- com.apple.dovecot.plist
- com.apple.mail.virus.plist
- com.apple.network.configurationchange.plist
- com.apple.softwareupdate.updateavailable.plist
- com.apple.timemachine.alerts.plist

On my 10.9 Server I have these 3 files in that directory.

- AdaptiveFirewall.plist
- DHABlock.plist
- HostBlockingLogic.plist

It looks like all of the files from 10.8 Server have been replaced in 10.9 Server by everything in this directory: /Applications/Server.app/Contents/ServerRoot/System/Library/Alerts

- Caching.bundle
- CertificateAlerts.bundle
- Common.bundle
- Disk.bundle
- Firewall.bundle
- Mail.bundle
- NetworkConfiguration.bundle
- ProfileManager.bundle
- SoftwareUpdate.bundle
- TimeMachine.bundle
- XcodeServer.bundle

10.8 also includes a file at this path.

/Applications/Server.app/Contents/ServerRoot/private/etc/emond.d/alert_mail_theme.mime

It is pretty obvious this is the template for sending emails. This file has been slightly changed and moved to the following path in 10.9 Server.

/Applications/Server.app/Contents/ServerRoot/System/Library/Alerts/Common.bundle/Contents/Resources/AlertsMailTheme.mime

On 10.9 Server I ran fs_usage and sent a test alert to see if emond is executed and it's not. Instead it looks like control goes from Server.app to servermgrd, to AlertsDaemon, then to sendmail.

This is one of the things that says to me that Apple is moving away from emond. If I had 10.11 Server I could check, but I'm guessing emond is totally unused.

Triggering events at startup, periodic.daily.midnight, and with the Mach service

Once emond reads the rule files it processes all rules with an event type of "startup". In some rule files you'll see this text.

```
 <key>eventTypes</key>
<array>
    <string>startup</string>
</array>
```

After these startup events are executed emond just sits and waits for a message or for a periodic event. The config file for emond has a key named "periodicEvents", which defines the periodic.daily.midnight event. It looks like this.

```
 <key>periodicEvents</key>
<array>
    <dict>
        <key>eventType</key>
        <string>periodic.daily.midnight</string>
        <key>startTime</key>
        <string>0</string>
    </dict>
</array>
```

Other than startup and periodic events, emond just sits and waits.

The launchd plist file for emond specifies a Mach service named com.apple.emond.evtq. I think "evtq" is short for "event queue." Anything should be able to send something directly to emond using this Mach service (too bad I don't know how to send messages to a Mach service with a script--I bet Python could do it, but I'm not interested enough to check).

emlog.pl

The script emlog.pl (10.11 version) is started by a launchd plist file (/System/Library/LaunchDaemons/com.apple.emlog.plist). The plist file specifies a socket listener on port 60762. Based on the contents of the script I am pretty sure that typically this script processes one line of text from system.log or secure.log and then quits. I have no idea what is reading the log files and sending the text to the script and don't much care.

The script checks to see if the line it is parsing matches some patterns and if it does then it constructs an event string and sends that to xssendevent. The event string is formated as ASCII plist. Here are some example event strings.

```
 $eventString = "{ eventType = auth.failure; eventSource = emlog.pl; eventDetails = {clientIP = \"$addr\"; hostPort = 21;
$eventString = "{ eventType = auth.failure; eventSource = emlog.pl; eventDetails = {clientIP = \"$address\"; protocolName
$eventString = "{ eventType = auth.failure; eventSource = emlog.pl; eventDetails = {username = \"$username\"; clientIP = \
$eventString = "{ eventType = auth.success; eventSource = emlog.pl; eventDetails = {username = \"$username\"; clientIP = \
$eventString = "{ eventType = network.probe; eventSource = emlog.pl; eventDetails = #{sourceIP = \"$address\"; port = 22;}
```

emlog.pl sends the event string to xssendevent using this perl code.

```
 open $OUTSTREAM, "|/usr/libexec/xssendevent" or die "Cannot launch /usr/libexec/xssendevent $!";
...
print $OUTSTREAM $eventString;
```

xssendevent

I am pretty sure xssendevent is one of the things that sends messages to the com.apple.emond.evtq Mach service. So xssendevent just reads stdin and I believe xssendevent turns the plist to an NSDictionary object and sends that to the Mach service com.apple.emond.evtq. This kind of makes xssendevent a command line bridge emond.

Event type

Once an event is sent to emond, I believe emond looks through all of the rules it has and finds any events that match the "eventType". I already mentioned the "startup" and "periodic.daily.midnight" events. The other event types are all defined in the event strings and the rule plist files. For example, emlog.pl creates an event with the following string.

```
 "{ eventType = auth.failure; eventSource = emlog.pl; eventDetails = {clientIP = \"$addr\"; hostPort = 21; protocolName =
```

The file /Applications/Server.app/Contents/ServerRoot/private/etc/emond.d/rules/AdaptiveFirewall.plist contains an event with that name "auth.failure" with this code.

```
 <key>eventTypes</key>
<array>
    <string>auth.failure</string>
</array>
```

So the event type names can really be anything you want, you just have to have an event match a rule. This system sounds a lot like (NSDistributedNotificationCenter) [https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSDistributedNotificationCenter_Cla

This is a list of some of the event types I've seen.

- auth.failure
- auth.success
- com.apple.network.suppress.notifications
- com.apple.network.suppress.notifications
- com.apple.xsan.fibreEvents
- com.apple.xsan.overQuota.group
- com.apple.xsan.overQuota.user
- com.apple.xsan.testNotification
- com.apple.xsan.volFreespace
- com.apple.xsan.volRestart
- security.action.host_blocked
- smtp.receive.badrecipient

Variables

Before I can talk about the next step you have to understand variables. Rules can define variables. Here is an example of a rule defined variable. This is what you'd see in the plist file for a rule.

```
 <key>variables</key>
<dict>
    <key>hostBlockThreshold</key>
    <integer>25</integer>
    <key>hostMinBlockTime</key>
    <integer>15</integer>
</dict>
```

I am pretty sure variables can also be defined by the eventDetails portion of the event string sent to xssendevent. For example, this event string from emlog.pl defines "clientIP", "hostPort", and "protocolName". I believe eventType is also turned into a variable.

```
 "{ eventType = auth.failure; eventSource = emlog.pl; eventDetails = {clientIP = \"$addr\"; hostPort = 21; protocolName =
```

There are also some builtin variables. Here are the 2 I've observed.

```
 builtin:hostName
builtin:now
```

You can also use the global keyword. This implies there is variable scope...

```
 global:lastSuppressionTime
```

Use a variable like this:

```
 ${variableName}
```

Here are real examples (including one named eventType).

```
 ${event:blockDuration}
${event:clientIP}
${event:eventTimestamp}
${event:eventType}
${event:hostAddress}
```

Since a rule plist file can have multiple rules (the root of the plist is an array instead of a dict) you can have rules that actually change variables that were previously defined. For example here is how one variable was incremented.

```
 <key>${old}</key>
<string>${old} + 1</string>
```

You can also do some other fancy stuff in the brackets.

```
 ${builtin:now-24:00:00.00}
${event:freePercent%.2f}
```

You can also load variables from a file

```
 <key>loadVariablesFromFile</key>
<string>/Library/Preferences/Xsan/notifications3.plist</string>
```

There are also a section in the emond.plist file for initialVariables.

Event criterion

Next, there is criterion. If a criterion exists, it must be true for the actions to be executed. Criterion are basically if statements formated as plist. You have an operator and operands. The operators I've seen are as follows.

- LessThan
- GreaterThan
- True
- Defined
- NotEmpty

There are probably more (like Equal or NotEqual) and you'd probably see them if you ran strings on the emond binary.

The operand is a variable name or, if you use the bracket notation, the value of the variable. Here are a few examples.

${FreeSpaceThreshold} > ${event:Percentage}

```
 <key>criterion</key>
<array>
    <dict>
        <key>operator</key>
        <string>GreaterThan</string>
```

```
        <key>operands</key>
        <array>
            <string>${FreeSpaceThreshold}</string>
            <string>${event:Percentage}</string>
        </array>
    </dict>
</array>
```

true

```
 <key>criterion</key>
<array>
    <dict>
        <key>operator</key>
        <string>True</string>
    </dict>
</array>
```

defined event:clientIP

```
 <key>criterion</key>
<array>
    <dict>
        <key>operator</key>
        <string>Defined</string>
        <key>operands</key>
        <array>
            <string>event:clientIP</string>
        </array>
    </dict>
</array>
```

global:notificationContacts != ""

```
 <key>criterion</key>
<array>
    <dict>
        <key>operator</key>
        <string>NotEmpty</string>
        <key>operands</key>
        <array>
            <string>global:notificationContacts</string>
        </array>
    </dict>
</array>
```

${${event:clientIP}-BadAuthCount} > ${hostBlockThreshold}

```
 <key>criterion</key>
<array>
    <dict>
```

```
        <key>operator</key>
        <string>GreaterThan</string>
        <key>operands</key>
        <array>
            <string>${${event:clientIP}-BadAuthCount}</string>
            <string>${hostBlockThreshold}</string>
        </array>
    </dict>
</array>
```

Putting if statements in plist form takes a whole lot more text than if you were to just write "if (something1 > something2)".

There is also this key. I'm guessing this turns the array of criterion into or's instead of and's.

```
 <key>allowPartialCriterionMatch</key>
```

Actions

Once an eventType matches and the criterion (if any) are met, then the actions are performed. I found these different actions.

- Log
- SendEmail
- SendSMS
- SendNotification
- RunCommand

This is what the log action looks like.

```
 <dict>
    <key>type</key>
    <string>Log</string>
    <key>message</key>
    <string>Host at ${event:clientIP} will be blocked for at least ${hostMinBlockTime} minutes</string>
    <key>facility</key>
    <string>AdaptiveFirewall</string>
    <key>logLevel</key>
    <string>Notice</string>
    <key>logType</key>
    <string>Syslog</string>
</dict>
```

Here is another log action that uses ASL instead of syslog.

```
 <dict>
    <key>type</key>
    <string>Log</string>
    <key>message</key>
    <string>${event:eventTimestamp} Host at ${event:hostAddress} was blocked for ${event:blockDuration}</string>
    <key>facility</key>
    <string>AdaptiveFirewall</string>
    <key>logLevel</key>
```

```
    <string>Warning</string>
    <key>logType</key>
    <string>ASL</string>
    <key>parameters</key>
    <dict>
        <key>eventType</key>
        <string>${event:eventType}</string>
        <key>hostAddress</key>
        <string>${event:hostAddress}</string>
    </dict>
</dict>
```

Send Email. Not all of these keys are required. I got by with just type, message, subject, and recipientAddresses.

```
 <dict>
    <key>type</key>
    <string>SendEmail</string>
    <key>message</key>
    <string>Please do not be alarmed. This is only a test for SAN ${event:SANName}.</string>
    <key>subject</key>
    <string>${event:SANName}: Test notification</string>
    <key>localizationBundlePath</key>
    <string>/usr/libexec/xsanmgr/bundles/xsanmgr_xsan.bundle</string>
    <key>relayHost</key>
    <string>${event:relayHost}</string>
    <key>adminEmail</key>
    <string>${event:adminEmail}</string>
    <key>recipientAddresses</key>
    <array>
        <string>${event:emailRecipients}</string>
    </array>
</dict>
```

Send SMS. I couldn't get this to work. If you really want to send an SMS you can use the SendEmail action and an email to text message service. Most carriers have their own and you can find many at www.emailtextmessages.com.

It is also worth mentioning that if you use a small carrier that rents from a larger carrier, use the larger carrier's service. Since some carriers rent from multiple larger carriers you might need to try several different large carriers or call them and ask what cell towers your phone connects to (it depends on the SIM card).

```
 <dict>
    <key>type</key>
    <string>SendSMS</string>
    <key>message</key>
    <string>Please do not be alarmed. This is only a test for SAN ${event:SANName}.</string>
    <key>localizationBundlePath</key>
    <string>/usr/libexec/xsanmgr/bundles/xsanmgr_xsan.bundle</string>
    <key>relayHost</key>
    <string>${event:relayHost}</string>
    <key>adminEmail</key>
    <string>${event:adminEmail}</string>
    <key>recipientAddresses</key>
    <array>
```

```
        <string>${event:smsRecipients}</string>
    </array>
</dict>
```

Send notification. The OS Notification Center?... I don't know (didn't feel like testing).

```
 <dict>
    <key>type</key>
    <string>SendNotification</string>
    <key>name</key>
    <string>EventMonitorNotification</string>
    <key>message</key>
    <string>EventMonitorNotification</string>
    <key>details</key>
    <dict>
        <key>hostBlockedTime</key>
        <string>${hostMinBlockTime}</string>
        <key>message</key>
        <string>HostBlocked</string>
    </dict>
</dict>
```

Just run a command.

```
 <dict>
    <key>type</key>
    <string>RunCommand</string>
    <key>command</key>
    <string>/System/Library/Filesystems/acfs.fs/Contents/bin/xsandaily</string>
    <key>user</key>
    <string>root</string>
    <key>group</key>
    <string>wheel</string>
    <key>arguments</key>
    <array>
        <string>-a</string>
        <string>${event:clientIP}</string>
        <string>-t</string>
        <string>${hostMinBlockTime}</string>
    </array>
</dict>
```

More reading

- There's a paragraph that mentions emond in the book (Mac OS X Security and Mobility v10.6: Using a Firewall) [http://www.peachpit.com/articles/article.aspx?p=1573022&seqNum=2]. The most interesting part of that paragraph is that it says, "emond is an off-limits subsystem". That makes me smile.

- There's also an old very interesting Apple discussion thread. In it keeperofthecheese says, "the Leopard release of this feature was intended to be Apple-internal, which is why this (pretty powerful, IMHO) feature is not yet widely used throughout the system." I'm pretty sure that refers to emond. This seems to indicate to me that emond was intended to be much more. I wonder if keeperofthecheese worked on emond or knew the person who did.

- Mac OS X Server Advanced Server Administration Version 10.6 Snow Leopard spends half of page 184 discussing emond in more detail than the man pages. It also says, "the file formats and settings in emond.conf and rules plists are not documented for customer use. Tampering could result in an unusable notification system and is unsupported."
- emond man page.
- emond.plist man page. The most interesting part is that it tells how to enable debugging and logging, which would probably help figure out a lot more than my documentation.
- emlog.pl man page.
- xssendevent man page.

Conclusion

So we aren't suppose to use emond and it looks like it's on the chopping block. But it's there for now and there is enough information to figure out how to use it if you want. I've been struggling with my own version of a log scanner that emails and texts and that's what finally motivated me to look at emond.

But seriously, after looking at it, I'm not too convinced it is even worth it. It appears to be a plist (data) driven system just to send emails, log messages, etc. Considering how easy it is to log (logger), run a command (system or ``), or send an email in a perl script (pipe to sendmail), I can't see why I'd want to use emond instead of a script. And the criterion section is just atrocious. This whole system could be replaced with a few dozen much more readable lines of Perl (or any other scripting language, even BASH). There's got to be more to this than I know about. In fact, as I write that, I remember the alert_mail_theme.mime file and that it is an email template. Ok, it would take more than a few dozen lines of Perl to duplicate that functionality. But I guess I don't need that much power.

I think for now all I'm really going to do is create a startup rule that sends me an email. That way I know when one of my servers restarts. Here it is.

```
 <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<array>
    <dict>
        <key>name</key>
        <string>Startup Email</string>
        <key>enabled</key>
        <true/>
        <key>eventTypes</key>
        <array>
            <string>startup</string>
        </array>
        <key>actions</key>
        <array>
            <dict>
                <key>type</key>
                <string>SendEmail</string>
                <key>message</key>
                <string>${builtin:hostName} started up.</string>
                <key>subject</key>
                <string>${builtin:hostName} started up.</string>
                <key>adminEmail</key>
                <string>root</string>
```

```
                <key>recipientAddresses</key>
                <array>
                    <string>james.reynolds@example.com</string>
                </array>
            </dict>
            <dict>
                <key>type</key>
                <string>SendEmail</string>
                <key>message</key>
                <string>${builtin:hostName} started up.</string>
                <key>subject</key>
                <string>${builtin:hostName} started up.</string>
                <key>adminEmail</key>
                <string>root</string>
                <key>recipientAddresses</key>
                <array>
                    <string>123-456-7890@txt.att.net</string>
                </array>
            </dict>
        </array>
    </dict>
</array>
</plist>
```

Of course, the server that I really needed to be notified when it restarts is running OS X 10.6 client, so it doesn't even have emond. Irony. Here's the perl script and launchd plist that I'm using for it.

```
 #!/usr/bin/perl -w

use strict;

chomp ( my $hostname = `hostname` );
send_mail (
    'to' => 'james.reynolds@example.com',
    'from' => 'root',
    'subject' => "$hostname started up.",
    'message' => "$hostname started up.",
);
send_mail (
    'to' => '123-456-7890@txt.att.net',
    'from' => 'root',
    'subject' => "$hostname started up.",
    'message' => "$hostname started up.",
);

sub send_mail {
    my %h = @_;
    open SENDMAIL, "|/usr/sbin/sendmail -oi -t" or die "/usr/sbin/sendmail: $!\n";
    print SENDMAIL << "EOF";
From: $h{'from'}
To: $h{'to'}
Subject: $h{'subject'}
```

```
$h{'message'}
EOF
    close SENDMAIL;
}
```

And the launchd plist.

```
 <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.magnusviri.startup_email</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/local/bin/startup_email.pl</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>LaunchOnlyOnce</key>
    <true/>
</dict>
</plist>
```

Anyway, back to work I guess and figure out how to get my logscanner to work.

Published: 2016-04-07, last edited: 2020-05-11,
