LogLogger for macOS Sierra
Introductory Tutorial and Reference
Howard Oakley https://eclecticlight.co

You'll have discovered by now that Sierra's new Console app doesn't do most of the things that the app used to in El Capitan. This is a brief introduction to using my free app LogLogger to get useful information from Sierra's logs. This is an introduction, which shows how to get the app started, and how to perform two of the commonest and most basic tasks. At the end is a short reference to its controls and features.

What you need

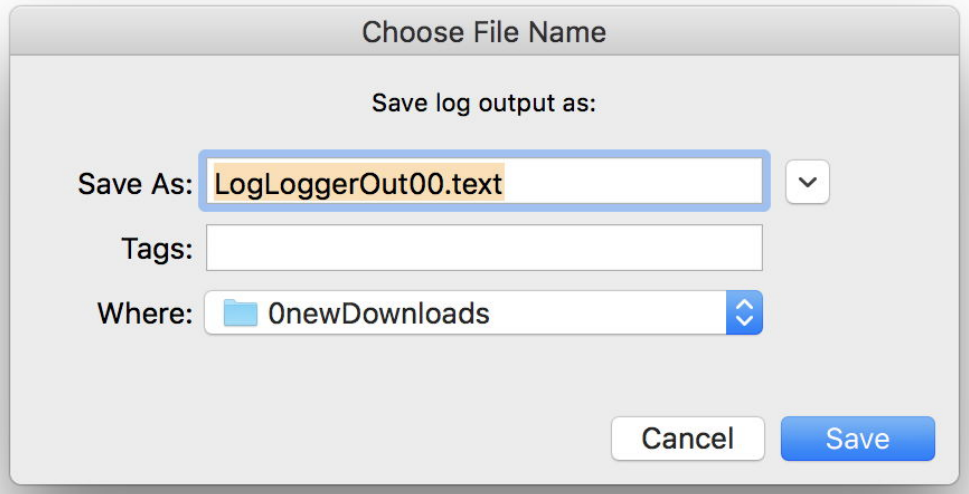
- A Mac running macOS Sierra 10.12 to 10.12.2. I recommend that you use 10.12.2, as that does fix a bug in the logging system which will help, and as far as I know, everything here should work on 10.12.3 and later too.
- A copy of the latest version of LogLogger from https://eclecticlight.co
  (This is delivered by secure HTTPS download.)

Getting started

LogLogger comes compressed as a Zip file, which you should decompress, and move the app to your preferred folder, such as /Applications. It is not fussy where it is run from, though.

LogLogger now uses my developer code signature, so should run normally first time after installation.

If it does not, you should still be able to use it safely. Instead of double-clicking to run it the first time, select the app icon in a Finder window, and use the Finder's Open command to run it. You will be prompted to confirm that you want to open it: click on the button to say that you do. After that first use, on that Mac, you should be able to use it normally. If you are told that it is damaged or the signature is incorrect, please contact me.

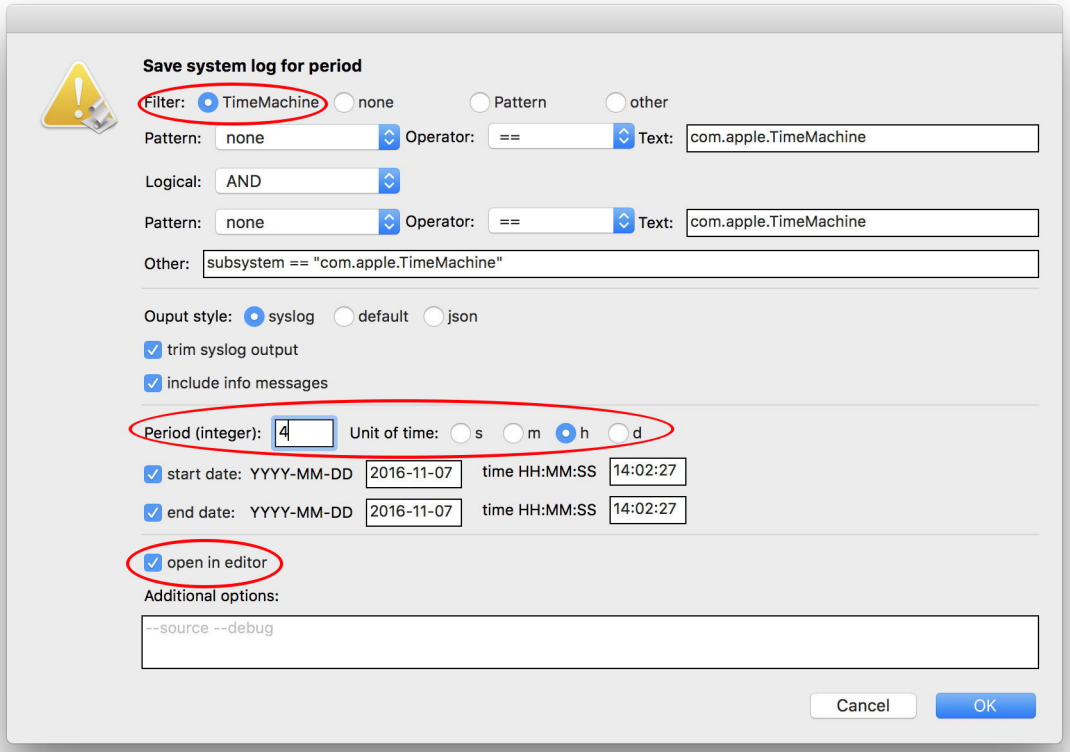


The first thing that LogLogger then does is to prompt you to save the first log excerpt, using a standard File Save dialog. If you want to navigate further, click on the disclose (down) arrow at the upper right.

LogLogger is designed to help you save a whole series of log excerpts when you are exploring your logs. By all means save them to whichever folder you wish, but I recommend that you use the suggested file names, as they will automatically increment from LogLoggerOut00.text to LogLoggerOut01.text, and so on. If you want to use your own custom file names, that auto-increment system should work if you only have a single period in the name, if the two characters before that period are 00 to begin with, and if the extension is left as .text (or .txt). It aims to be a bit more flexible too, but those should be safe rules.

Checking that Time Machine backups are completing without errors

One of the commonest reasons for wanting to access your logs is to ensure that your backups are being made without problems. This is extremely simple and efficient using LogLogger. Once you have provided the name and location of the log excerpt file to save, in the File Save dialog, LogLogger's main dialog will be displayed.



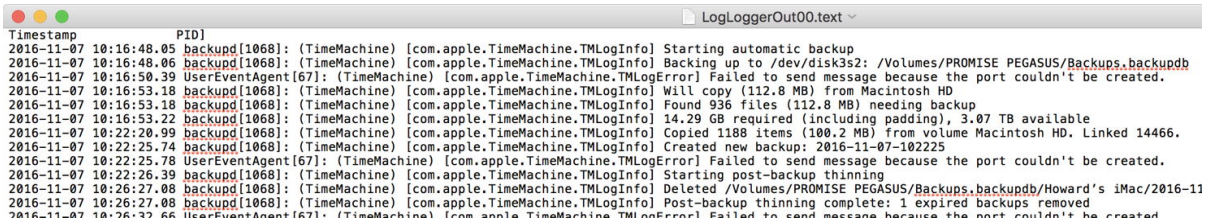
There are three important controls which you need to check.

At the top, the *Filter:* radio button should be set to the first option, *TimeMachine.*

Halfway down, the *Period (integer)* text box should be set to the number of hours (of hourly backups) that you want to inspect. Normally 2 to 4 are sufficient; the greater this period, the longer this will take. On that same line, set the *Unit of time* to **h** for hours, rather than its default of **s** for seconds.
Almost at the bottom of the dialog, check the box to *open in editor,* so that once the log excerpt has been created, it will automatically be opened using TextEdit (or your chosen default text editor).

Then click on the *OK* button.

LogLogger then assembles the shell command using `log show`, and executes it. There will be a short pause of several seconds while the log excerpt is prepared, saved to the file, and opened for you to inspect.



You can then resize TextEdit's window to check through your backup entries for errors. They should read something like:

```
2016-11-07 13:19:49.23 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Starting automatic backup
2016-11-07 13:19:49.58 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Backing up to /dev/disk3s2: /Volumes/PROMISE
PEGASUS/Backups.backupdb
2016-11-07 13:19:50.69 UserEventAgent[67]: (TimeMachine)
[com.apple.TimeMachine.TMLogError] Failed to send message because the port couldn't
be created.
2016-11-07 13:19:53.34 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Will copy (662.3 MB) from Macintosh HD
2016-11-07 13:19:53.34 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Found 658 files (662.3 MB) needing backup
2016-11-07 13:19:53.37 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] 14.95 GB required (including padding), 3.07 TB
available
2016-11-07 13:23:04.13 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Copied 931 items (642.1 MB) from volume Macintosh
HD. Linked 9886.
2016-11-07 13:23:09.47 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Created new backup: 2016-11-07-132308
2016-11-07 13:23:09.51 UserEventAgent[67]: (TimeMachine)
[com.apple.TimeMachine.TMLogError] Failed to send message because the port couldn't
be created.
2016-11-07 13:23:10.39 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Starting post-backup thinning
2016-11-07 13:23:10.39 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] No post-backup thinning needed: no expired
backups exist
2016-11-07 13:23:10.42 backupd[1553]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Backup completed successfully.
```

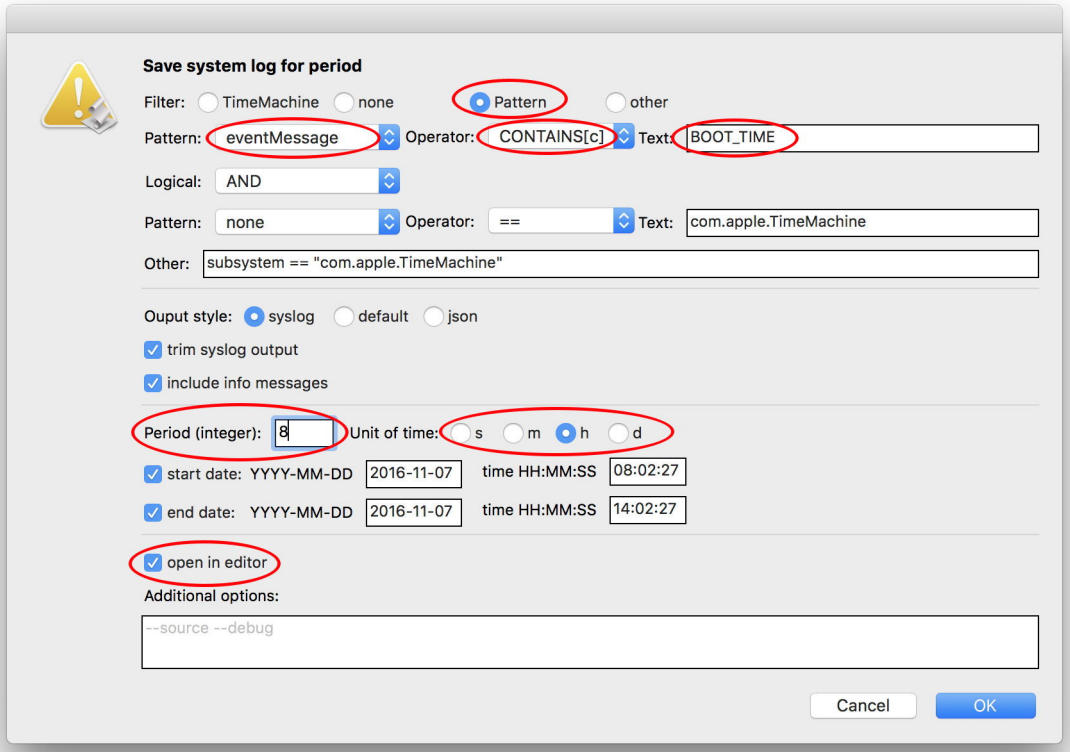
repeated every hour or so. Note the "Failed to send message" entry above refers to an irrelevant issue (and a bug in macOS).

Meanwhile, LogLogger will have gone back to its File Save dialog. As you're done now, just click on its *cancel* button, and the app will quit.

Checking a restart (or startup)

When your Mac starts up, it invariably makes a log entry to mark the start of that process, which contains the text `BOOT_TIME`. This is invaluable for identifying the exact time of each startup or restart, so that you can check out any problems with them.

To locate log entries containing `BOOT_TIME`, we're going to put that in as a filter pattern or predicate which will only return those lines containing that text.



Run LogLogger again, and complete the File Save dialog as before. In the main dialog, ensure the following controls are set.

At the top, set the radio button to *Pattern,* so that we can enter those details.

In the line below that, set the first popup menu to read *eventMessage,* as `BOOT_TIME` appears in the message part of the log entry. Set the next popup to *CONTAINS(c),* which looks for a match without being case-sensitive (although this will work with plain *CONTAINS,* which *is* case-sensitive). In the final text entry box, type `BOOT_TIME` being careful to use the underscore character, shift-hyphen.

Although you can search for this in a specific period defined by *start* and *end,* you will get more hits if you use a time period before the present. For that, set the *Period (integer)* to several hours, a period within which you know your Mac has started up. Then set the *Unit of time* to **h** for hours.

Just above the bottom of the dialog, select the *open in editor* checkbox, so that the log excerpt will be opened for you to view. Then click on the *OK* button.

After a period of a few seconds, the log excerpt will open in TextEdit, and show one line for each startup which occurred in the specified time period. In my case, that reads:

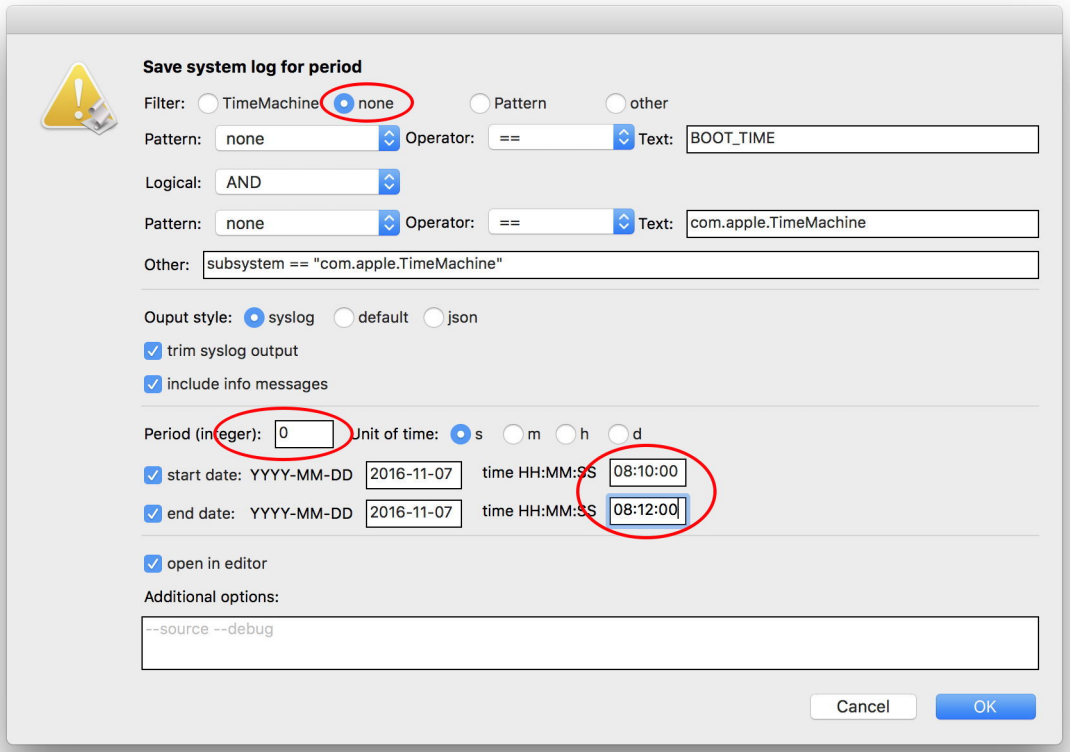
```
Timestamp PID]
2016-11-07 08:10:20.85 syslogd[66]: (libsystem_c.dylib) BOOT_TIME:
1478506214 0
```

So we now know that the last startup was getting underway at just after 8:10 local time today.

Viewing all log entries

Instead of quitting LogLogger by cancelling the next dialog, we'll use that startup time to see just how many entries are made in the log around a startup or restart.

In the File Save dialog, agree to the next log excerpt being made and saved to the next filename. Then in the main dialog, we'll set up to get a complete log extract for just two minutes around `BOOT_TIME`.



At the top, set the radio button to *none,* which turns all filtering off, and will save every log entry to our text file.

Set the *Period (integer)* to **0** (zero), which means that *start* and *end* times will be used instead.

Ensure the *start date* line is checked (enabled), and that today's date has been automatically entered for the start date. Then in that same row, set the *time* to a few seconds before the time of the `BOOT_TIME`, as shown in the last log excerpt, which should still be open in TextEdit.

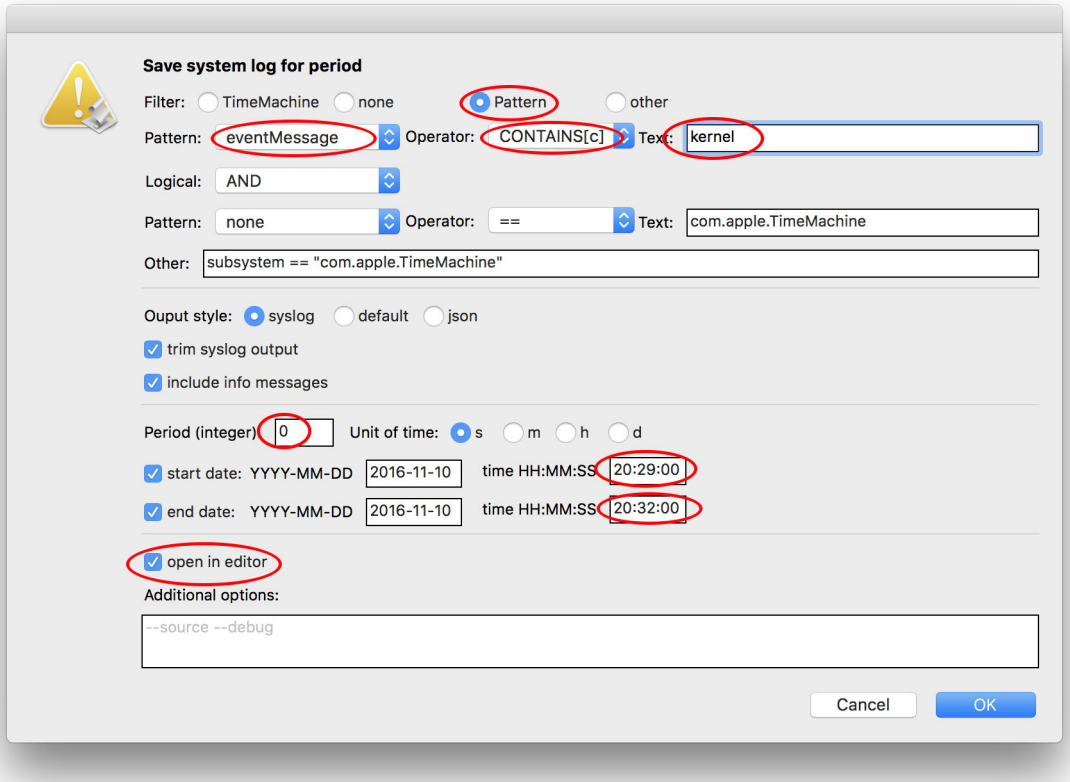
Ensure the *end date* line is also checked, that today's date has been automatically entered, and then set the *end* time to just two minutes after the start time, which should be after the `BOOT_TIME` entry. So if your `BOOT_TIME` was 11:10:30, set the start time to, say, 11:10:10, and the end time to 11:12:10.

This log excerpt is going to be very large, so you may prefer to turn *open in editor* off – that is up to you. Then click on the *OK* button.

Here, my resulting log extract for just those two minutes contains over 127,000 entries, and is 17.6 MB in size. Unless you are seriously bored, you will not want to browse that: looking for problems with an extension, or other details, would be a nightmare. So the next section looks at filter predicates which you can use to hone in on problems, to investigate them.

Use broad filters to hone in on problems

Complete the File Save dialog to store the next log excerpt, and then run a broad filter to work out what you need to look at in more detail. One good filter setting picks up all messages containing `kernel`, which is good if you are looking at a system-level problem:



Set LogLogger's main dialog up like this, filtering on *eventMessage* which *CONTAINS(c)* the text `kernel` (case insensitive). Ensure that you set the *Period (integer)* back to **0** in order to use *start date* and *end date* below, then adjust the time settings to start the excerpt a little

before the `BOOT_TIME`, and end it two or three minutes later, to catch the whole of startup. You'll also want to open the excerpt in your text editor for immediate viewing.

You may find that you need to adjust the start and end times a little after this, but from now on all the settings below the top section will remain unchanged, so I omit them from the screenshots below.

If you have got your times right, the first line in the excerpt should tell you which version of the kernel is running, a useful piece of information in itself:

```
2016-11-10 20:29:58.75 kernel[0]: Darwin Kernel Version 16.1.0: Thu Oct 13
21:26:57 PDT 2016; root:xnu-3789.21.3~60/RELEASE_X86_64
```

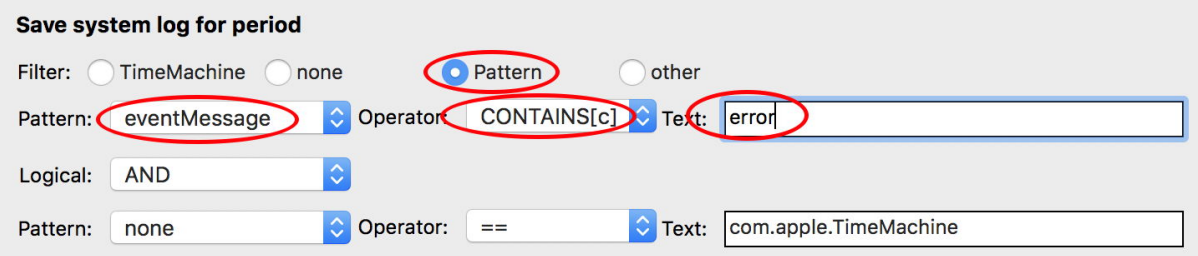
(that's for Sierra 10.12.1).

Shortly after that, you'll see two lines

```
2016-11-10 20:30:02.51 DumpPanic[88]: Attempting to extract kernel core if
it exists
2016-11-10 20:30:02.51 DumpPanic[88]: HandleKernelCore(): SadMac partition
not found
```

which confirm that the restart did not occur after a kernel panic. If it does, then the information from that should be recovered and made available.
Lower down are various housekeeping entries, and you'll probably see some long messages about `SandboxViolation`. Otherwise, if you start seeing other entries, they may well correspond to a problem.

Another good broad filter is to look for messages containing errors:



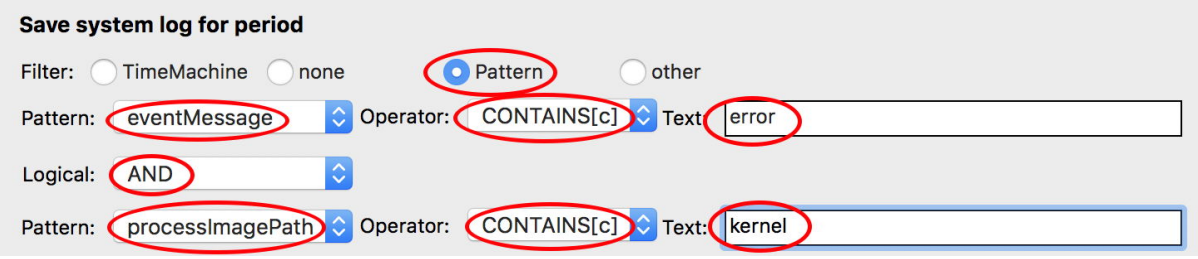
A lot of these will relate to security systems, like

```
2016-11-10 20:30:03.04 amfid[142]: (Security)
[com.apple.securityd.security_exception] MacOS error: -67050
```

As is usual in Unix, success is an error code of 0, so you will also see reports of processes completing successfully! If you want to decode Apple's sprawling error numbers, an excellent site to use is https://www.osstatus.com

More specific filters

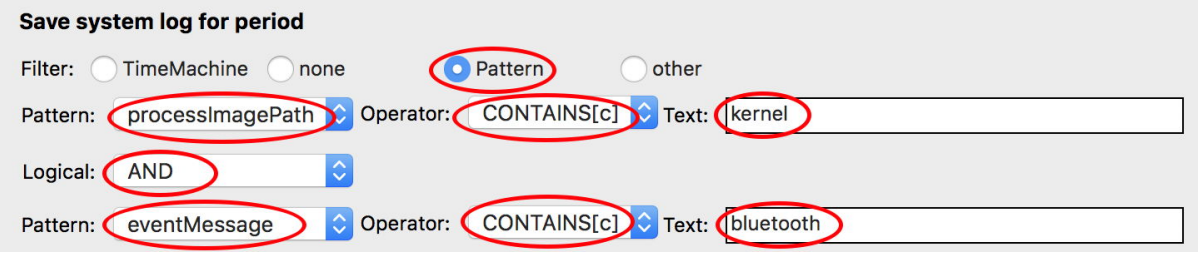
Sometimes, you'll need to narrow your filter by adding a second pattern. There's a huge range of these which might prove useful, so I will just offer a few suggestions here.



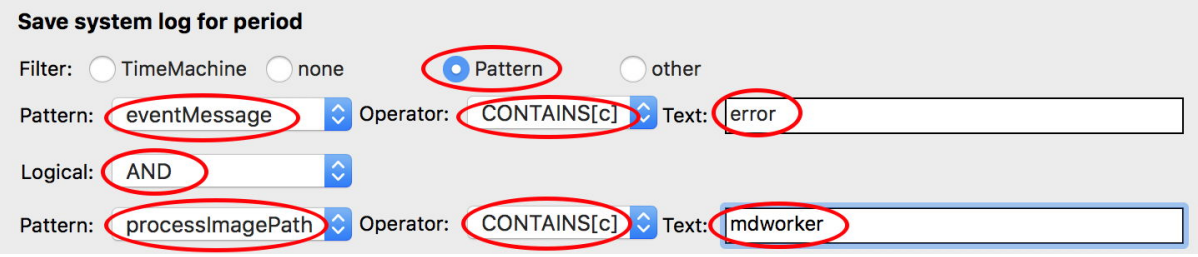
Try looking at error messages returned from specific processes - here the kernel. This excerpt should be very short, such as:

```
2016-11-10 20:30:00.18 kernel[0]: (IO80211Family) init: error getting
PHY_MODE; using MODE_UNKNOWN
2016-11-10 20:30:29.19 kernel[0]: utun_start: ifnet_disable_output returned
error 12
```

Any more than that and you'll want to follow them up. You may then want to broaden the time period that you are examining, something which makes sense when you are using narrow filters.



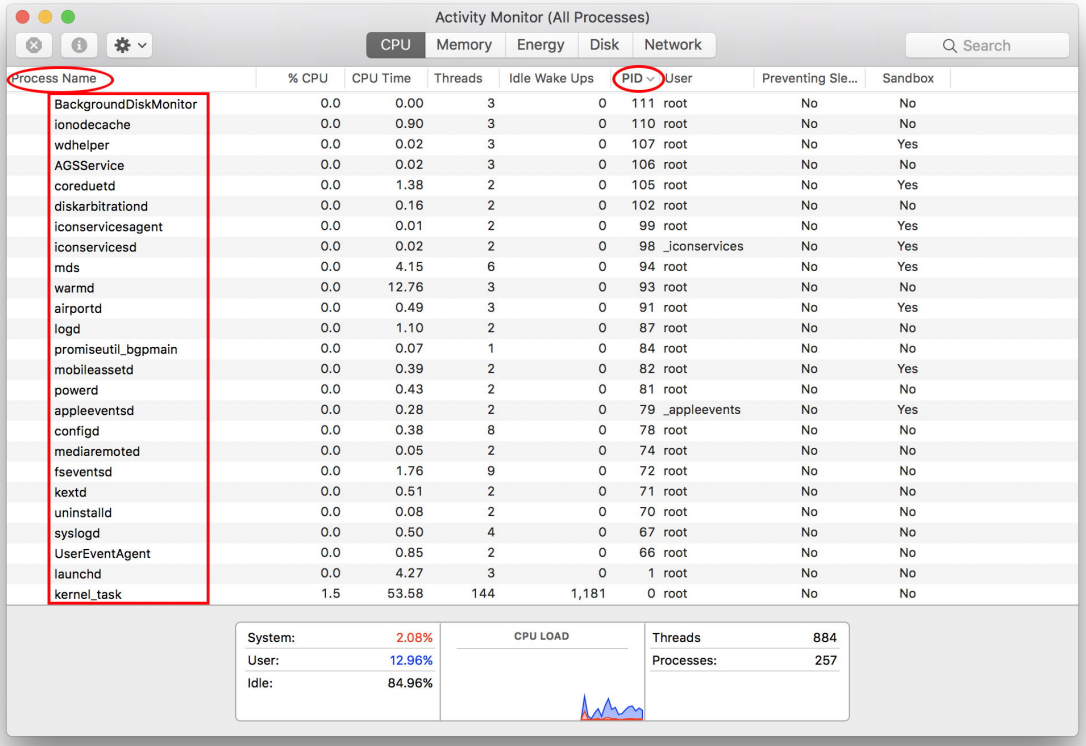
This combination of messages from the kernel which contain reference to Bluetooth is excellent for looking at Bluetooth problems: you can track Bluetooth starting up, and becoming fully functional.



This looks at a common problem which can start soon after startup, repeatedly crashing of `mdworker` processes, which index metadata for Spotlight search.

Discovering processImagePath names

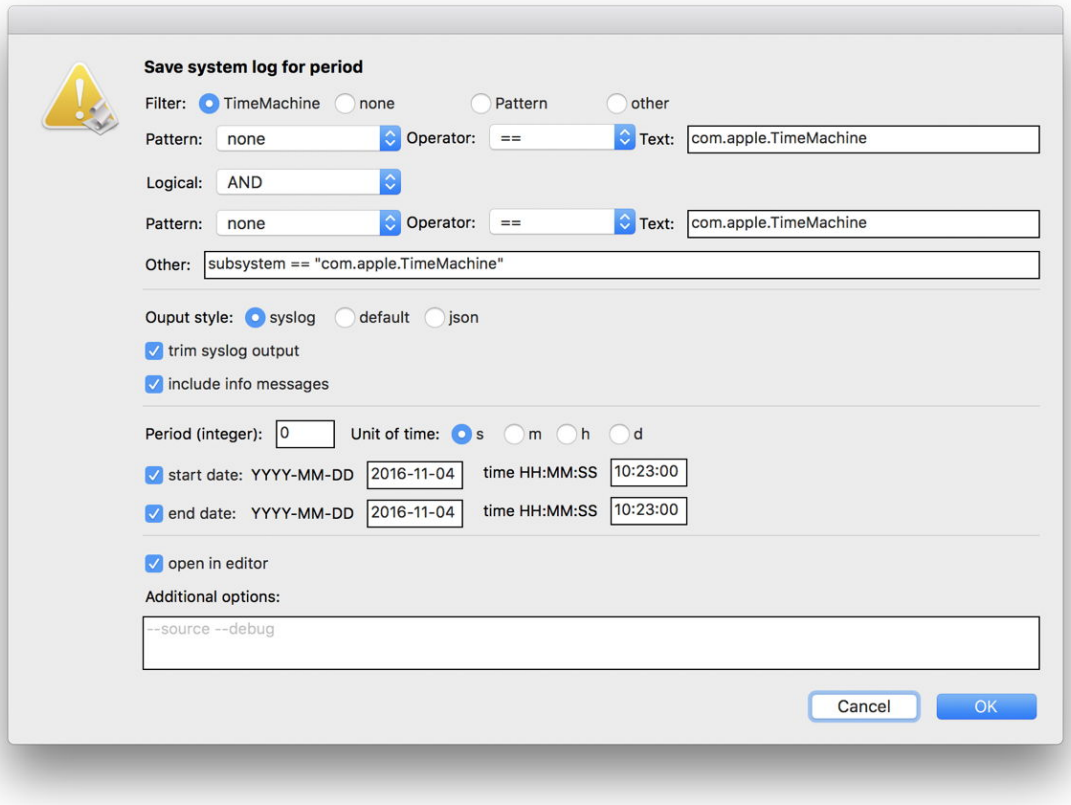
Although log messages, as set using *eventMessage,* can usually be guessed quite easily, you may not know many process names which can be entered as *processImagePath.* Thankfully your Mac contains an excellent listing of the names of all those active processes at any time.



Start Activity Monitor, set it to show CPU usage according to the tools at the top, and the items listed in the first column, *Process Name,* are excellent candidates for use when filtering according to *processImagePath.* You can make this even easier if you click on the top of the *PID* column, as those processes with the lowest numbers are started soonest after startup: the kernel (named here `kernel_task`) has the PID of 0, and `launchd` (which launches all subsequent processes) has the PID of 1.

Try to take some time exploring the last startup on your Mac, using LogLogger and its log excerpts. They'll help you understand what goes on during a normal startup, so you'll find it easier to recognise the abnormal.

Reference Guide to LogLogger's Main Dialog



The first section sets any predicates to be used to **filter** the entries to be included in its output. If you just want *Time Machine* entries, leave the button set to that. If you want all log entries (beware: for any length of time the output file will be huge), set it to *none.*

The other two radio buttons require you to enter predicate information below.

Pattern lets you create one or two filter terms using the popup menus and text boxes below. If you select this, you must configure at least the first *Pattern* line.

The two lines which start with *Pattern:* let you build the most popular filter expressions. If you set both, then the *Logical* operator (by default AND) will be applied to combine them.

The first popup menu in each of the two *Pattern* lines determines what is examined in the filter. On offer are:

- *eventMessage* – for this, you specify a text pattern, or text, within the message, or an activity name.
- *processImagePath* – this matches the text pattern in the name of the process which originated the event.
- *senderImagePath* – this matches the text pattern in the name of the sender, which might be the name of a library, extension, or executable.

- *subsystem* – this matches the subsystem specifier, e.g. `com.apple.TimeMachine`. Although potentially valuable, subsystems are not yet widely used, and discovering which is which is not easy. Use with caution.

The *Operator* popup menu in each of the two *Pattern* lines determines what the filter actually does. Operators available include:

- == is the equality operator, as in == "com.apple.TimeMachine"
- *!=* is the inequality operator
- *BEGINSWITH* is for text which begins with the quoted text, and is case- and diacritic-sensitive
- *CONTAINS* is for text which contains the quoted text, and is case- and diacritic-sensitive
- *CONTAINS[c]* is for text which contains the quoted text, and is case-**in**sensitive and diacritic-sensitive
- *ENDSWITH* is for text which ends with the quoted text, and is case- and diacritic-sensitive

Logical operators which can be used to combine two filter patterns include:

- *AND* which is simple, logical AND - both patterns are true
- *OR* is simple, logical OR - either pattern is true
- *AND (NOT* is logical and, but the second pattern is NOT true
- *OR (NOT* is logical or, but the second pattern is NOT true.

For example, shortly after restarting from a freeze I ran the app to show all the kernel entries from just before the freeze until the current moment. To do that, I set up just the first *Pattern* line, to read

```
processImagePath CONTAINS[c] kernel
```

Note that when you use the `log` command in Terminal, you must insert text in quotation marks " ". In the final text boxes for *Patterns,* do **not** use quotation marks unless they are part of the search string: LogLogger automatically puts the contents of the *Text* boxes into quotation marks when it builds the command.

other allows you to enter any other valid predicate which you wish, such as that shown by default in the *Other* box below: here you need to give the full predicate, including any " " for text, which will simply be placed inside single quotes ' ' and prefaced by `--predicate`

The middle section concerns the **style and formatting** of the output. The standard is to use traditional system log style, similar to the previous Console app. You will probably want that with the *trim* feature turned on, to make the lines more compact. The *default* style is based on the new logs' content, which is much more extensive and detailed. You will want to turn trimming off in that case. The final option for *JSON* format is valuable if you want to read the log output into another app which takes JSON format; don't use trim with that, or it will become a real mess.

Normally you should *include info* messages: this does make a difference.

The third section concerns the **period** of logs to cover. There are two ways to specify this: a period of time up to the present, or start and end times, and the app now provides for both. If you enter a non-zero positive integer into the *Period* text box, then the log excerpt will cover that period up to the present moment. If you leave the period set to zero (0), then the start and end times will be used instead.

With a positive integer entered into the *Period* box, you need to set the *Unit of time,* which defaults to seconds for safety. Units are selected from seconds, minutes, hours, or days.

Start and *end* dates are initially set to today's date and the time that you start the app. Ensure that both their checkboxes are checked (or they will be omitted, and possibly result in a vast log excerpt), and edit the dates and times in the standard format. Using other formats will almost certainly result in errors, and unexpected results.

The final section allows you to open the resulting file in your default text editor (usually TextEdit), and add any other text you want to the `log show` command. If you leave that text box blank (the default), then no additional text will be added to the command, which is the normal way to run LogLogger.

If you click on the *OK* button, after a few seconds or longer, the requested log excerpt should then be saved into your specified text output file, and opened in your text editor if you opted for that to happen. If you click on the *Cancel* button, LogLogger will quit.

If you use the standard syslog style with trimming, the first line will normally be junk, thereafter it will look something like

```
2016-10-12 18:19:34.49 backupd[10374]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Starting automatic backup
2016-10-12 18:19:34.77 backupd[10374]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Backing up to /dev/disk3s2: /Volumes/PROMISE
PEGASUS/Backups.backupdb
2016-10-12 18:19:35.60 UserEventAgent[66]: (TimeMachine)
[com.apple.TimeMachine.TMLogError] Failed to send message because the port couldn't
be created.
2016-10-12 18:19:37.66 backupd[10374]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Will copy (75.4 MB) from Macintosh HD
2016-10-12 18:19:37.67 backupd[10374]: (TimeMachine)
[com.apple.TimeMachine.TMLogInfo] Found 573 files (75.4 MB) needing backup
```

and so on.

At the end of the output file, after a blank line, LogLogger writes out the command line which it ran to generate the log output in the file above. You may find this useful if you encounter a problem, or to learn how to use the `log show` command yourself.

Having clicked on the *OK* button and obtained your log excerpt, the app then returns to present you with the file save dialog again, to create another log extract. This loop will continue until you *either* cancel on the file save dialog, or cancel on the main app dialog.