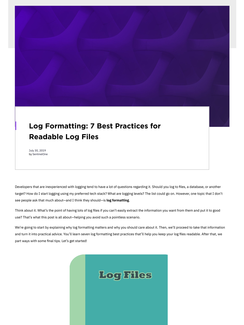


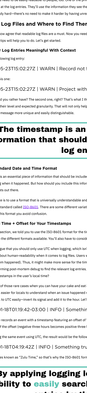
Log Formatting: 7 Best Practices for Readable Log Files

July 30, 2019
by SentinelOne

Developers that are inexperienced with logging tend to have a lot of questions regarding it. Should you log to files, a database, or another target? How do I start logging using my preferred tech stack? What are logging levels? The list could go on. However, one topic that I don't see people ask that much about—and I think they should—is **log formatting**.

Think about it. What's the point of having lots of log files if you can't easily extract the information you want from them and put it to good use? That's what this post is all about—helping you avoid such a pointless scenario.

We're going to start by explaining why log formatting matters and why you should care about it. Then, we'll proceed to take that information and turn it into practical advice. You'll learn seven log formatting best practices that'll help you keep your log files readable. After that, we part ways with some final tips. Let's get started!



Why Should You Care About Log Formatting?

Log formatting is crucial to achieving readable log files. But why would you need readable (i.e., human-readable) log files in the first place?

You might argue that a log file should be machine-readable. I whole-heartedly agree with that. Having machine-readable logs enables you to run tools that can automatically parse, process, and analyze them. As a result, you can gain valuable insights that would otherwise be lost to you.

But your log files need to be easily readable to people, too. One of the primary purposes of logging is to enable post-mortem debugging. People will look at the log files. They'll also read the information they see there to try and reconstruct what happened with the application. That job is already hard—there's no need to make it harder by having unreadable log files.

Fantastic Log Files and Where to Find Them: 4 Log Formatting Best Practices

Hopefully, you now agree that readable log files are a must. Now you need to know how to achieve readable log files, and that's what the following seven tips will help you do. Let's get started.

1. Make Your Log Entries Meaningful With Context

Consider the following log entry:

```
2017-05-23T15:02.27Z | WARN | Record not found
```

Consider now this one:

```
2017-05-23T15:02.27Z | WARN | Project with the id '53' was not found
```

Which one would you rather have? The second one, right? That's what I thought. Add as much context as possible to your log messages with respect to their level and expected granularity. That will not only help its readers to get a headstart on fixing the problem. But it will also make each message more unique and easily distinguishable.

> **The timestamp is an essential piece of information that should be included in every log entry.**

2. Use a Standard Date and Time Format

The timestamp is an essential piece of information that should be included in every log entry. It's no use to know that something happened without knowing when it happened. But how should you include this information? That can be a tricky decision since there are many date/time formats out there.

The best practice is to use a format that is universally understandable and free from ambiguities. The good news is that such format exists and it's an ISO standard called ISO 8601. There are some different variations of the format, but its advantages are present in all of them. In short, by using this format you avoid confusion.

3. Use Local Time + Offset for Your Timestamps

In the previous section, we told you to use the ISO 8601 format for the timestamps on your log entries. But the difficulty of handling time is not restricted to the different formats available. You'll also have to consider timezones and Daylight Saving Time (DST).

Some people argue that you should use UTC when logging, which isn't bad advice (it does solve the problems above.) But today we're mainly talking about human-readability when it comes to log files. Users will talk in terms of their timezone (e.g., it was about 2pm EST when the problem happened). Thus, it might make more sense for the timestamps to be in users' timezone. That makes it easier for the developer performing post-mortem debug to find the relevant log entries. But you can also solve the DST and timezones problems, while keeping the timestamps in UTC, at least to a level.

Well, that's one of those rare cases when you can have your cake and eat it too. Just use the local time, along with the offset from UTC. That way, it becomes easier for locals to understand when an issue happened relative to their own time. The offset allows you to convert the timestamp back to UTC easily—invert its signal and add it to the hour. Let's look at an example.

```
2019-01-18T01:19:42-03:00 | INFO | Something truly awesome happened.
```

The entry above records an event with a timestamp featuring an offset of "-03:00" hours from UTC. To get the time in UTC, we'll have to invert the sign of the offset (negative three hours becomes positive three hours) and add that to the time. The result would be "04:19:42."

If we were to log the same event using UTC, the result would be the following:

```
2019-01-18T04:19:42Z | INFO | Something truly awesome happened.
```

UTC is sometimes known as "Zulu Time," as that's why the ISO 8601 format uses a "Z" to express time in UTC.

> **By applying logging levels, you gain the ability to easily search and filter your log entries by their levels.**

4. Employ Logging Levels Correctly

Properly employing logging levels is one of the most important things you can do for your log files. That'll make it easier for automatic parsers to understand them. But it also helps human readers, which is our main focus today. The correct use of levels will make your log files more easily searchable and, yes, readable. But what are logging levels?

Briefly, logging levels are labels. You use them to categorize your log entries by urgency, or severity. By applying logging levels, you gain the ability to easily search and filter your log entries by their levels. Levels also help you control the granularity of your log entries—more on that later.

So, what are the best practices regarding logging levels?

First, always add the log level to your log entries. Additionally, make sure to display the level in uppercase (ERROR instead of "error.") That will make the level easily distinguishable from the other information on the entry. Finally, use the appropriate level for each event. For instance, don't use ERROR for a typical, expected event—for example, if a user logged on the system.

5. Split Your Logging to Different Targets Based on Their Granularity

This section is sort of a continuation of the previous one since it is related to logging levels. And why is that? Besides enabling searching/filtering, logging levels also allow you to control the granularity of your logs easily. If you're debugging, you'll probably want to log almost everything to nail down the cause of whatever bug you're trying to fix.

When it comes to production, on the other hand, it makes sense to log fewer events. For instance, you might log the actions of the user that are important from a business logic perspective, but not each interaction the user had with the UI. For instance, you would usually log "User X created a new project template" instead of "User X opened the new project template screen, User X clicked on the "Add new template" button, …" and so on.

So, the advice here is to log to different targets, based on level/granularity. "Target" here means the log's destination. It might be a database table, a text file, and so on. When you're searching through a log, trying to understand what went wrong in production, you don't want to be bogged down by thousands of debugging entries.

6. Include the Stack Trace When Logging an Exception

This section should be short since its title pretty much says everything there is to it. When logging an exception, you should always include its stack trace. For the developer performing a post-mortem debug, the stack trace is essential information that will help them connect the dots.

7. Include the Name of the Thread When Logging From a Multi-Threaded Application

Like the previous section, this one also has its contents spelled out by its title. When logging from an application with multiple threads, always include the name of the thread where the event happened. That will enable searching/filtering, making the entries not only easily searchable but also more readable for humans.

Log Formatting Is Good for Machines and Great for Humans

Logging's vital for any non-trivial applications. Without logging, it'd be extremely hard to understand how the code we write behaves "in the wild." We'll have a hard time trying to fix bugs. We'll have no idea about how our users actually use the software we write. In short—we'll be clueless to the extreme.

However, it's not enough just to have logging in place. Getting started with logging is the first step. After that, you need to tackle new concepts. You must ensure that your logs are readable, both by machines and humans. If you have log files that are well-formatted and easily parseable, you can use the tools at your disposal to process, manage, and analyze them. That way, you can learn from your logs, gaining insights about your application and your users that would otherwise be lost.

By having log files that are easy for humans to read, you make your incident debug less of a burden for you and your fellow developers. Log formatting represents a win-for-everyone, so start employing it ASAP.

Like this article? Follow us on LinkedIn, Twitter, YouTube or Facebook to see the content we post.

Read more about Cyber Security

- Blackout Speed to Truth

SentinelOne
©2019 SentinelOne. All Rights Reserved.
Privacy Policy | Terms and Conditions