Architecture

Architecture of the lumberjack framework.

Introduction

If you're familiar with other logging frameworks (such as the popular log4j) then you'll quickly recognize the concepts. If logging is new to you, that's OK because lumberjack was designed to be simple to use.

If you haven't already done so, you may want to read the getting started page.

Overview

At the heart of the framework is the DDLog file. This file provides the various DDLog macros that will replace your NSLog statements. For example:

```
DDLogWarn(@"Specified file does not exist");
```

During compilation these macros get expanded to something akin to:

```
if(LOG_WARN) /* Execute log statement */
```

What this means is that the compiler has the ability to automatically prune log statements that will never get executed. (For more information see the getting started page. You may also choose to use dynamic log levels.)

When a log statement is executed, DDLog forwards the log message to every logger.

Loggers

A logger is a class that does something with a log message. The lumberjack framework comes with several different loggers. (You can also create your own.) Loggers such as DDASLLogger and DDTTYLogger can be used to duplicate the functionality of NSLog. And DDFileLogger can be used to write log messages to a log file.

You can have multiple simultaneous loggers. So you could, for example, log to the console and a file at the same time. You can add and remove loggers at any time.

When your application first starts, the lumberjack framework has no loggers. So you'll want to add at least one logger if you want your log statements to go anywhere. (For information on how to add a logger, see the getting started page.)

Loggers may have certain settings that you can configure. For example, the DDFileLogger has options for controlling how often to roll log files.

All loggers will also support an optional formatter.

Formatters

Formatters allow you to format a log message before the logger logs it. For example, there is no need to add a timestamp to normal console log messages because the console automatically does that. However, if you are logging to a file, you would likely want to prepend a timestamp to the log message.

So you could add a formatter to DDFileLogger to automatically prepend a timestamp to every log message.

Formatters can also be used to filter log messages.

And remember that formatters are applied individually to loggers. So you can format and/or filter on a per logger

And remember that formatters are applied individually to loggers. So you can format and/or filter on a per-logger basis.

Learn how to write your own custom formatters.