GettingStarted

Getting started with the lumberjack framework.

Introduction

There are 3 steps to getting started with the logging framework:

1. Add the lumberjack files to your project.
2. Configure the framework.
3. Convert your NSLog statements to use the Lumberjack macros

Add the lumberjack files to your project

The main files you need to add are:

- DDLog (Basis of entire framework)
- DDASLLogger (sends log statements to Apple System Logger, so they show up on Console.app)
- DDTTYLogger (sends log statements to Xcode console - if available)
- DDFileLogger (sends log statements to a file)

DDLog is mandatory, and the others are optional depending on how you intend to use the framework. For example, if you don't intend to log to a file, you can skip DDFileLogger. Or if you want to skip ASL in favor of faster file logging, you could skip DDASLLogger.

Configure the framework

One of first things you'll want to do in your application is configure the logging framework. This is normally done in the applicationDidFinishLaunching method.

A couple lines of code is all you need to get started:

```
[DDLog addLogger:[DDASLLogger sharedInstance]];
[DDLog addLogger:[DDTTYLogger sharedInstance]];
```

This will add a pair of "loggers" to the logging framework. In other words, your log statements will be sent to the Console.app and the Xcode console (just like a normal NSLog).

Part of the power of the logging framework is its flexibility. If you also wanted your log statements to be written to a file, then you could add and configure a file logger:

```
fileLogger = [[DDFileLogger alloc] init];
fileLogger.rollingFrequency = 60 * 60 * 24; // 24 hour rolling
fileLogger.logFileManager.maximumNumberOfLogFiles = 7;

[DDLog addLogger:fileLogger];
```

The above code tells the application to keep a week's worth of log files on the system.

Convert your NSLog statements to DDLog

The DDLog header file defines the macros that you will use to replace your NSLog statements. Essentially they look like this:

```
// Convert from this:
NSLog(@"Broken sprocket detected!");
```

```
NSLog(@"User selected file:%@ withSize:%u", filePath, fileSize);

// To this:
DDLogError(@"Broken sprocket detected!");
DDLogVerbose(@"User selected file:%@ withSize:%u", filePath, fileSize);
```

As you can see, the **DDLog macros have the exact same syntax as NSLog**.

So all you need to do is decide which log level each NSLog statement belongs to. By default, there are 4 options available:

- DDLogError
- DDLogWarn
- DDLogInfo
- DDLogVerbose

(You can also customize the levels or the level names. Or you can add fine-grained control on top of or instead of simple levels.)

Which log level you choose per NSLog statement depends, of course, on the severity of the message.

These tie into the log level just as you would expect

- If you set the log level to LOG_LEVEL_ERROR, then you will only see DDLogError statements.
- If you set the log level to LOG_LEVEL_WARN, then you will only see DDLogError and DDLogWarn statements.
- If you set the log level to LOG_LEVEL_INFO, you'll see Error, Warn and Info statements.
- If you set the log level to LOG_LEVEL_VERBOSE, you'll see all DDLog statements.
- If you set the log level to LOG_LEVEL_OFF, you won't see any DDLog statements.

Where do I set the log level? Do I have to use a single log level for my entire project?

Of course not! We all know what it's like to debug or add new features. You want verbose logging just for the part that you're currently working on. The lumberjack framework gives you per file debugging control. So you can change the log level on just that file you're editing.

(Of course there are many other advanced options, such as a global log level, per xcode configuration levels, per logger levels, etc. But we'll get to that in another article.)

Here's all it takes to convert your log statements:

```
// CONVERT FROM THIS

#import "Sprocket.h"

@implementation Sprocket

- (void)someMethod
{
    NSLog(@"Meet George Jetson");
}

@end

// TO THIS

#import "Sprocket.h"
#import "DDLog.h"

static const int ddLogLevel = LOG_LEVEL_VERBOSE;
```

```
@implementation Sprocket

- (void)someMethod
{
    DDLogVerbose(@"Meet George Jetson");
}

@end
```

Notice that the log level is declared as a constant. This means that DDLog statements above the log level threshold will be compiled out of your project (when your compiler has optimizations turned on, as it would for your release build).

This is just the tip of the iceberg.

Find out how to:

- Automatically use different log levels for your debug vs release builds
- Dynamically change the log level at runtime
- Tailor the log levels to suite your needs
- Write your own custom loggers
- Upload log files to your server