# DynamicLogLevels

*Dynamically changing log levels during run-time*

## Introduction

When you define your log level, you generally define it in a manner similar to this:

```
static const int ddLogLevel = LOG_LEVEL_WARN;
```

What this means is that your log level is declared as a constant. It cannot be changed.

This has the advantage that the compiler can automatically prune DDLog statements above the log level threshold during compilation.
However, it also has the disadvantage that you cannot change your log level during run-time.

To allow a dynamic log level, all we need to do is remove the "const" part:

```
static int ddLogLevel = LOG_LEVEL_WARN;
```

This means that we can change the log level when/how we want. For example, maybe we're debugging some specific part of our application.

```
- (void)startTask
{
    ddLogLevel = LOG_LEVEL_VERBOSE;
    [self startTaskInBackground];
}

- (void)taskDidFinish
{
    ddLogLevel = LOG_LEVEL_WARN;
}
```

You will incur a tiny little performance penalty for using dynamic log levels. (A single integer comparison... gasp!) However, due to the architecture and speed of the lumberjack framework, most DDLog statements will **still be faster than NSLog**.
(For more information, see the performance page.)

### Crazy Awesomeness

Dynamic logging, as used in the example above, can be helpful for debugging.

**But what if we wanted to take it to the next level?**

Imagine if you could alter log levels via the !NSUserDefaults system...

A user is complaining about the application not acting properly (in the preference pane somewhere). So you simply tell them to issue a

```
defaults write com.yourapp.prefsLogLevel 4
```

And in your preference pane code, you have something like this:

```
static int ddLogLevel = LOG_LEVEL_WARN;

+ (void)initialize
{
    NSNumber *logLevel = [[NSUserDefaults standardUserDefaults] objectForKey"@prefsLogLevel"];
    if (logLevel)
        ddLogLevel = [logLevel intValue];
}
```

When they relaunch the app, your preferences UI now has verbose logging, and the user can send you the log files.

**But what if we wanted to take it to the next level after that?**

Imagine a preference screen somewhere in your application (maybe hidden, maybe in preferences, maybe accessible via an embedded http server) that lists every single file in your application with a dynamic log level. With a click of the button you can change the log level of any file!

Just imagine: You're at a party and a friend comes up to you and says, "Hey, love the latest version of your app... But this part isn't working properly anymore." You take a look and you're not quite sure what's going on. No problem. You just toggle the log level of the buggy source code file (without restarting the app), and it immediately starts dumping all kinds of juicy debug info to the log file. You grab the log file, inspect it the next day at the office, and quickly identify the problem.

The hypothetical situation might be a bit silly, but you get the idea. Some bugs only appear on certain machines/devices. Some bugs only appear when Xcode isn't available. And some bugs mysteriously disappear when you restart the application. So it would be helpful if we could dynamically change the log level when the problem appears and as the application is still running.

How to design a user interface for something like this is entirely up to you. But the lumberjack framework can help out with the plumbing!

One of the tasks involved in such an endeavor is to create a list of all the source code files that support dynamic logging. Depending on the size of your project, this may be annoying. Not to mention that as your project grows, you would need to keep this list up-to-date. Plus, you would have to import the header of every single file that supports dynamic logging... And each header file would have to declare methods to get and set the log level.... It's just a big headache. Luckily it's a headache that can completely be avoided due to the dynamic nature of the objective-c runtime!

The lumberjack framework has something called "registered dynamic logging". Here's all you have to do in your source code files:

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

static int ddLogLevel = LOG_LEVEL_WARN;

@implementation Sprocket

+ (int)ddLogLevel
{
        return ddLogLevel;
}

+ (void)ddSetLogLevel:(int)logLevel
{
        ddLogLevel = logLevel;
}

// The rest of your code...

@end
```

In other words, just add the two class methods (ddLogLevel and ddSetLogLevel:).

Now take a look at the registered dynamic logging section of DDLog:

```
/**
 * Registered Dynamic Logging
 *
 * These methods allow you to obtain a list of classes that are using registered dynamic logging,
 * and also provides methods to get and set their log level during run time.
**/

+ (NSArray *)registeredClasses;
+ (NSArray *)registeredClassNames;

+ (int)logLevelForClass:(Class)aClass;
+ (int)logLevelForClassWithName:(NSString *)aClassName;

+ (void)setLogLevel:(int)logLevel forClass:(Class)aClass;
+ (void)setLogLevel:(int)logLevel forClass:(Class)aClass;
+ (void)setLogLevel:(int)logLevel forClassWithName:(NSString *)aClassName;
```

So if you call `[DDLog registeredClasses]` , it would return an array that contains the Sprocket class.

And if you call `[DDLog logLevelForClass:[Sprocket class]]` , it will invoke and return `[Sprocket ddLogLevel]` .

What this means is that you can easily enumerate through all the classes (or class names) that use registered dynamic logging. You don't have to keep a big list somewhere. Or even import all the header files. You can just enumerate a list, and the lumberjack framework takes care of the rest.