

# Logging in iOS Applications

## Overview

This tutorial provides the required code snippets in order to add logging capabilities in iOS applications.

**Prerequisite:** Make sure to read the overview of client-side log collection (../).

**Note:** Using `OCLogger` in Swift requires creating an `OCLogger` extension class (this class can be a separate Swift file, or an extension to your current Swift file):

```
extension OCLogger {
    //Log methods with no metadata

    func logTraceWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_TRACE, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    func logDebugWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    func logInfoWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_INFO, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    func logWarnWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_WARN, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    func logErrorWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    func logFatalWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    func logAnalyticsWithMessages(message:String, _ args: CVarArgType...) {
        logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
    }

    //Log methods with metadata

    func logTraceWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
```

```

    logWithLevel(OCLogger_TRACE, message: message, args:getVaList(args), userInfo:userInfo)
}

func logDebugWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args), userInfo:userInfo)
}

func logInfoWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_INFO, message: message, args:getVaList(args), userInfo:userInfo)
}

func logWarnWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_WARN, message: message, args:getVaList(args), userInfo:userInfo)
}

func logErrorWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args), userInfo:userInfo)
}

func logFatalWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args), userInfo:userInfo)
}

func logAnalyticsWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args), userInfo:userInfo)
}
}

```

After including the extension class you may now use `OCLogger` in Swift.

## Enabling log capture

By default, log capture is enabled. Log capture saves logs to the client, and can be enabled or disabled programmatically. Logs are sent to the server with an explicit send call, or with auto log.

**Note:** Enabling log capture at verbose levels can impact the consumption of the device CPU, file system space, and the size of the payload when the client sends logs over the network.

To disable log capturing:

### Objective-C

```
[OCLogger setCapture:NO];
```

### Swift

```
OCLogger.setCapture(false);
```

## Sending captured logs

Send logs to the MobileFirst according to your application's logic. Auto log send can also be enabled to automatically send logs. If logs are not sent before the maximum size is reached, the log file is then purged in favor of newer logs.

**Note:** Adopt the following pattern when you collect log data. Sending data periodically ensures that you are seeing your log data in near real-time in the MobileFirst Analytics Console.

### Objective-C

```
[NSTimer scheduledTimerWithTimeInterval:60
target:[OCLogger class]
selector:@selector(send)
userInfo:nil
repeats:YES];
```

### Swift

```
var timer = NSTimer.scheduledTimerWithTimeInterval(60,
target:OCLogger.self,
selector: #selector(OCLogger.send),
userInfo: nil,
repeats: true)
```

To ensure that all captured logs are sent, consider one of the following strategies:

- Call the `send` method at a time interval.
- Call the `send` method from within the app lifecycle event callbacks.
- Increase the max file size of the persistent log buffer (in bytes):

### Objective-C

```
[OCLogger setMaxFileSize:150000];
```

### Swift

```
OCLogger.setMaxFileSize(150000);
```

## Auto log sending

By default, auto log send is enabled. Each time a successful resource request is sent to the server, the captured logs are also sent, with a 60-second minimum interval between sends. Auto log send can be enabled or disabled from the client. By default auto log send is enabled.

### Objective-C

To enable:

```
[OCLogger setAutoSendLogs:YES];
```

To disable:

```
[OCLogger setAutoSendLogs:NO];
```

## Swift

To enable:

```
OCLogger.setAutoSendLogs(true);
```

To disable:

```
OCLogger.setAutoSendLogs(false);
```

## Fine-tuning with the Logger API

The MobileFirst client-side SDK makes internal use of the Logger API. By default, you are capturing log entries made by the SDK. To fine-tune log collection, use logger instances with package names. You can also control which logging level is captured by the analytics using server-side filters.

## Objective-C

As an example of capturing logs only where the level is `ERROR` for the `myApp` package name, follow these steps.

1. Use a `logger` instance with the `myApp` package name.

```
OCLogger *logger = [OCLogger getInstanceWithPackage:@"MyApp"];
```

2. **Optional:** Specify a filter to restrict log capture and log output to only the specified level and package programmatically.

```
[OCLogger setFilters:@{@"MyApp": @(OCLogger_ERROR)}];
```

3. **Optional:** Control the filters remotely by fetching a server configuration profile.

## Swift

1. Using the extension as explained in the Overview, create a logger instance for your package.

```
let logger : OCLogger = OCLogger.getInstanceWithPackage("MyTestLoggerPackage");
```

2. **Optional:** Specify a logging level.

```
OCLogger.setLevel(OCLogger_DEBUG);
```

3. **Optional:** Control the filters remotely by fetching a server configuration profile.

## Fetching server configuration profiles

Logging levels can be set by the client, or by retrieving configuration profiles from the server. From the MobileFirst Operations Console, a log level can be set globally (all logger instances) or for a specific package or packages. For information on configuring the filter from the MobileFirst Operations Console,

see [Configuring log filters \(../../analytics/console/log-filters/\)](#). For the client to fetch the configuration from the server, the `updateConfigFromServer` method must be called from a place in the code that is regularly run, such as in the app lifecycle callbacks.

## Objective-C

```
[OCLogger updateConfigFromServer];
```

## Swift

```
OCLogger.updateConfigFromServer();
```

# Logging example

Outputs to a browser JavaScript console, LogCat, or Xcode console.

## Objective-C

```
#import "OCLogger.h"
+ (int) sum:(int) a with:(int) b{
    int sum = a + b;
    [OCLogger setLevel:DEBUG];
    OCLogger* mathLogger = [OCLogger getInstanceWithPackage:@"MathUtils"];
    NSString* logMessage = [NSString stringWithFormat:@"sum called with args %d and %d. Returning %d", a, b, sum];
    [mathLogger debug:logMessage];
    return sum;
}
```

## Swift

```
func sum(a: Int, b: Int) -> Int{
    var sum = a + b;
    let logger = OCLogger.getInstanceWithPackage("MathUtils");

    logger.logInfoWithMessages("sum called with args /(a) and /(b). Returning /(sum)");
    return sum;
}
```

*Last modified on*