

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 (../).

Logging example

Logs are outputted to the 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

Using OCLogger in Swift requires creating an OCLogger extension class (this class can be a separate swift file or an extension on 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.

```
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;  
}
```

Additional API Methods For Specific Tasks

Log capture is enabled by default. To turn log capture on or off:

Objective-C: objc

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

Swift: swift

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

The default capture level is FATAL in development and in production. To control the capture level (verbosity):

Objective-C: objc

```
[OCLogger setLevel:DEBUG];
```

Swift: swift

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

For more information about the `Logger` API, see the API reference in the user documentation.