Flurry Analytics

iOS SDK Instructions

SDK version 6.2.0
Updated: 2/17/2015

Welcome to Flurry Analytics!

This file contains:

1. Introduction

The Flurry iOS Analytics Agent allows you to track the usage and behavior of your iOS application on users' phones for viewing in the Flurry Analytics system. It is designed to be as easy as possible with a basic setup complete in under 5 minutes.

Please note that this SDK will only work with Xcode 4.5 or above. If you need an SDK for an older Xcode version please email support.

Flurry Agent does not require CoreLocation framework and will not collect GPS location by default. Developers who use their own CLLocationManager can set GPS location information in the Flurry Agent (see Optional Features for more information).

There are additional folders for use with Flurry Ads. These optional libraries provide alternate streams of revenue for your apps. If you would like to use Flurry Ads please refer to FlurryAds-iOS-README.pdf.

2. Integration

1. In the finder, drag Flurry/ into project's file folder. *(NOTE: If you are upgrading the Flurry iOS SDK, be sure to remove any existing Flurry library folders from your project's file folder before proceeding.)*

2. Now add it to your project:
File > Add Files to “Your Project” … > Flurry
- Destination: select Copy items into destination group’s folder (if needed)
- Folders: Choose 'Create groups for any added folders'
- Add to targets: select all targets that the lib will be used for

3. Add *Security.framework* to your app.

4. Add *SystemConfiguration.framework* to your app. This is required for Reachability to manage network operations efficiently.

5. In your Application Delegate:
   - Import Flurry and inside "application:didFinishLaunchingWithOptions:"
     add: [Flurry startSession:@"YOUR_API_KEY"];

Objective - C

```
#import "Flurry.h"

- (BOOL)application:(UIApplication *)application
 didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

   [Flurry startSession:@"YOUR_API_KEY"];
   //your code
}
```

You're done! That's all you need to do to begin receiving basic metric data.

To integrate with Swift applications:

* In your Swift application, add a new Objective-C .m file to your project.
* When prompted with creating a bridge header file approve the request.
* Remove the unused .m file.
* In the bridge header file import the Flurry.h header file.
* In your application's AppDelegate.swift file, inside application didFinishLaunchingWithOptions add the Flurry startSession call.

Swift Code :

```
func application(application:UIApplication, didFinishLaunchingWithOptions
launchOptions: NSDictionary) -> Bool {

    Flurry.startSession("YOUR_API_KEY")
   //your code
   return true
}
```

This stackoverflow article elaborates on the steps required to use Objective-C classes in Swift applications:
http://stackoverflow.com/questions/24002369/how-to-call-objective-c-code-from-swift/24005242#24005242

Note On Advanced Features

Flurry strongly recommends that you read the Advanced Features section below. These features are optional only because they require configuration on your part. It is most often the case that users of Flurry Analytics find as much or more value in the

Advanced Features listed below as in the features enabled by the above steps. Examples include:

- Crash Analytics - Review analytics and stack traces for the errors and crashes that occur in your app.
- Custom Events - Understand the interaction between your users and specific areas of functionality in the app.
- Demographics - Analyze and segment your users by demographic group based on the data they share with your app.

Be sure to read the *Flurry iOS Advertising README* as well to learn how you can monetize your app by showing ads to your users that are targeted using the data from Flurry Analytics.

3. Advanced Features

You can use the following methods to report additional data.

Tracking User Behavior

Utilizing Custom Events in your app, you can track the occurrence and state of most actions taken by users or even your app itself. Custom Events support functionality within Flurry Analytics such as Funnels, Segments, User Paths and others. To gain a better understanding of the value of Custom Events, see the video walkthrough available here.

Note that each of the logEvent APIs will return a status code that will indicate if the event was successfully recorded or not.

```
[Flurry logEvent:@"EVENT_NAME"];
```

Use *logEvent* to count the number of times certain events happen during a session of your application. This can be useful for measuring how often users perform various actions, for example. Your application is currently limited to counting occurrences for 300 different event ids (maximum length 255 characters) across all its sessions. When deciding on events to track consider adding dynamic parameters to capture various incarnations of an event (rather than creating multiple dynamic events)

```
[Flurry logEvent:@"EVENT_NAME" withParameters:YOUR_NSDictionary];
```

Use this version of logEvent to count the number of times certain events happen during a session of your application and to pass dynamic parameters to be recorded with that event. Event parameters can be passed in as a NSDictionary object (immutable copy) where the key and value objects must be NSString objects. For example, you could record that a user used your search box tool and also dynamically record which search terms the user entered. Your application is currently limited to counting occurrences for 100 different event ids (maximum length 255 characters) during one application session. Maximum of 10 event parameters per event is supported. There is limit of 1000 events in total for a session (for instance, you can have 100 different event ids each occurring up to 10 times during the application session) .

An example NSDictionary to use with this method could be:

```
NSDictionary *dictionary =
[NSDictionary dictionaryWithObjectsAndKeys:@"your dynamic parameter value",
                                           @"your dynamic parameter name",
                                           nil];
```

```
[Flurry logEvent:@"EVENT_NAME" timed:YES];
```

Use this version of *logEvent* to start timed event.

```
[Flurry logEvent:@"EVENT_NAME" withParameters:YOUR_NSDictionary timed:YES];
```

Use this version of *logEvent* to start timed event with event parameters.

```
[Flurry endTimedEvent:@"EVENT_NAME" withParameters:YOUR_NSDictionary];
```

Use *endTimedEvent* to end timed event before app exits, otherwise timed events automatically end when app exits. When ending the timed event, a new event parameters NSDictionary object can be used to update event

parameters. To keep event parameters the same, pass in nil for the event parameters NSDictionary object.

```
[Flurry logAllPageViewsForTarget:navigationController];
```
To enable Flurry agent to automatically detect and log page view, pass in an instance of *UINavigationController* or *UITabBarController* to *countPageViews*. Flurry agent will create a delegate on your object to detect user interactions. Each detected user interaction will automatically be logged as a page view. Each instance needs to only be passed to Flurry agent once. Multiple *UINavigationController* or *UITabBarController* instances can be passed to Flurry agent. This method along with the corresponding *stopLogPageViewsForTarget* replaces the deprecated *logAllPageViews* routine.

```
[Flurry stopLogPageViewsForTarget:navigationController];
```
Stops logging page views on a navigation controller that was previously observed with *logAllPageViewsForTarget*. Call this method before the instance of the navigation controller observed with *logAllPageViewsForTarget* is released.

```
[Flurry logPageView];
```
In the absence of *UINavigationController* and *UITabBarController*, you can manually detect user interactions. For each user interaction you want to manually log, you can use *logPageView* to log the page view.

Tracking Application Errors

```
[Flurry logError:@"ERROR_NAME" message:@"ERROR_MESSAGE" exception:e];
```
Use this to log exceptions and/or errors that occur in your app. Flurry will report the first 10 errors that occur in each session. The message parameter has been deprecated as of release 4.2.2 and the passed in message will not be viewable on the Flurry developer portal.

For the following features, please call these APIs before calling *[Flurry startSession:@"YOUR_API_KEY"]*:

Tracking Demographics

```
[Flurry setUserID:@"USER_ID"];
```
Use this to log the user's assigned ID or username in your system after identifying the user.

```
[Flurry setAge:21];
```
Use this to log the user's age after identifying the user. Valid inputs are 0 or greater.

```
[Flurry setGender:@"m"];
```
Use this to log the user's gender after identifying the user. Valid inputs are *m* (male) or *f* (female)

Tracking Location

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
[locationManager startUpdatingLocation];

CLLocation *location = locationManager.location;
[Flurry setLatitude:location.coordinate.latitude
          longitude:location.coordinate.longitude
 horizontalAccuracy:location.horizontalAccuracy
   verticalAccuracy:location.verticalAccuracy];
```

This allows you to set the current GPS location of the user. Flurry will keep only the last location information. If your app does not use location services in a meaningful way, using CLLocationManager can result in Apple rejecting the app submission.

Controlling Data Reporting

```
[Flurry setSessionReportsOnCloseEnabled:(BOOL)sendSessionReportsOnClose];
```
This option is enabled by default. When enabled, Flurry will attempt to send session data when the app is exited as

well as it normally does when the app is started. This will improve the speed at which your application analytics are updated but can prolong the app termination process due to network latency.

`[Flurry setSessionReportsOnPauseEnabled:(BOOL)sendSessionReportsOnPause];`
This option is enabled by default. When enabled, Flurry will attempt to send session data when the app is paused as well as it normally does when the app is started. This will improve the speed at which your application analytics are updated but can prolong the app pause process due to network latency.

`[Flurry setSecureTransportEnabled:(BOOL)secureTransport];`
This option is disabled by default. When enabled, Flurry will send session data over SSL when the app is paused as well as it normally does when the app is started. This has the potential to prolong the app pause process due to added network latency from secure handshaking and encryption.

`[Flurry setBackgroundSessionEnabled:(BOOL)backgroundSessionEnabled];`
This option is disabled by default. When enabled, Flurry will not finish the session if the app is paused for longer than the session expiration timeout. The session report will not be sent when the application is paused and will only be sent when the application is terminated. This allows for applications that run in the background to keep collecting events data. The time application spends in the background contributes to the length of the application session reported when the application terminates.

`[Flurry pauseBackgroundSession];`
This method is useful if *setBackgroundSessionEnabled:* is set to *YES*. It can be called when application finishes all background tasks (such as playing music) to pause the session. A session report is sent if *setSessionReportsOnPauseEnabled* is set to *YES*. If the app is resumed before the session expiration timeout, the session will continue, otherwise a new session will begin.

Crash Reporting
`[Flurry setCrashReportingEnabled:(BOOL)crashReportingEnabled];`

Please note: This call must be made prior to calling *[Flurry startSession:@"YOUR_API_KEY"]*.

This option is off by default. Flurry has disabled the Crash Reporting functionality to ensure that Flurry Crash Analytics feature does not affect the use of other crash reporting tools that might be in use by the application. When enabled, Flurry will collect crash reports and send it in the session data. The errors that are logged using the Flurry library will include stack traces that are captured at the point when the error is logged. Note that when this feature is enabled Flurry installs an uncaught exception handler and registers for signals. We strongly recommend that developers do not install any uncaught exception handlers in their app if they enable this feature. More information about the feature is available here.
Crash reporting is only supported on armv7, armv7s and armv8 architectures. If an application is built using multiple architecture slices then crash reporting will work on armv7 & above devices but will be disabled on armv6 devices.

Debug and Diagnostics
`[Flurry setLogLevel:(BOOL) logLevel];`
This is an optional method that can be used to set the level of Flurry SDK logs to be displayed on the console. The default log level is FlurryLogLevelCriticalOnly.

`[Flurry setDebugLogEnabled:(BOOL)debugLogEnabled];`
This option is off by default. When disabled the log level is set to FlurryLogLevelCriticalOnly. When enabled the log level is set to FlurryLogLevelDebug. This routine invokes setLogLevel.

`[Flurry getFlurryAgentVersion];`
This routine retrieves a string that contains the Flurry SDK agent version number and release version and is useful information that the developers can use for investigations.

`[Flurry addOrigin:(NSString *) withVersion:(NSString *)];`
This optional method is used by 3rd party products that wrap Flurry SDK. This method allows declaration of the Flurry SDK wrapper as the origin of the Flurry sessions. As a general rule you should capture all the origin info related to your wrapper for Flurry SDK after every session start.

`[Flurry addOrigin:(NSString *) withVersion:(NSString *) withParameters:(NSDictionary *)];`
This optional method is used by 3rd party products that wrap Flurry SDK. This method allows declaration of the Flurry SDK wrapper as the origin of the Flurry sessions. As a general rule you should capture all the origin info related to your wrapper for Flurry SDK after every session start.

5. FAQ

How much does the Flurry Analytics SDK add to my app size?

The Flurry SDK will typically add 150 KB to the final app size.

When does the Flurry Agent send data?

By default, the Flurry Agent will send the stored metrics data to Flurry servers when the app starts, pauses, resumes, and terminates. To override default Agent behavior, you can turn off sending data on termination by adding the following call before you call `startSession`:
`[Flurry setSessionReportsOnCloseEnabled:NO];`

You can turn off sending data on pause by adding the following call before you call `startSession`:
`[Flurry setSessionReportsOnPauseEnabled:NO];`

How much data does the Agent send each session?

All data sent by the Flurry Agent is sent in a compact binary format. The total amount of data can vary but in most cases it is around 2Kb per session.

What data does the Agent send?

The data sent by the Flurry Agent includes time stamps, logged events, logged errors, and various device specific information. This is the same information that can be seen in the custom event logs on in the Event Analytics section. We do not collect personally identifiable information.

Does the Agent support iOS OS 3.x?

Yes, this version is a fat binary that includes slices for armv6, armv7, armv7s, arm64, i386 and x86_64. Support is provided for iOS 3.1 to iOS 8.x.

What version of XCode is required?

The Flurry SDK will support Xcode 4.5 and above. Please email support if you need to use older versions of the Flurry SDK.

Does this version collect the iOS UDID?

This version of the Flurry iOS SDK does not collect the iOS UDID.

Please let us know if you have any questions. If you need any help, just email iphonesupport@flurry.com!

Cheers,
The Flurry Team
http://www.flurry.com
iphonesupport@flurry.com