Flurry Analytics

iOS SDK Instructions

SDK version 8.0.1
Updated: 04/04/2017

Welcome to Flurry Analytics!

This file contains:

1. Introduction
2. Integration
3. Advanced Features
4. Watch Analytics
5. Revenue Analytics
6. FAQ

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 7 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:"
   - Create a sessionBuilder to set up all the pre-session properties
     FlurrySessionBuilder* builder = [FlurrySessionBuilder new];
   - Call the startSession API
     [Flurry startSession::@"YOUR_API_KEY" withSessionBuilder:builder];

Objective - C

```
#import "Flurry.h"

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

      FlurrySessionBuilder* builder = [[[[FlurrySessionBuilder new]
                withLogLevel:FlurryLogLevelAll]
                withCrashReporting:YES]
                withSessionContinueSeconds:10]
                withAppVersion:@"0.1.2"];

      [Flurry startSession:flurryAPIKey withSessionBuilder:builder];
   //your code
}
```

Also, the pre-session calls on this link http://flurrydev.github.io/FlurryiOSSDK6xAPI/interface_flurry.html need to be updated with the new calls.

- (FlurrySessionBuilder*) withAppVersion:(NSString *)value;

- (FlurrySessionBuilder*) withSessionContinueSeconds:(NSInteger)value;

- (FlurrySessionBuilder*) withCrashReporting:(BOOL)value;

- (FlurrySessionBuilder*) withLogLevel:(FlurryLogLevel) value;

- (FlurrySessionBuilder*) withShowErrorInLog:(BOOL) value;

- (FlurrySessionBuilder*) withTVSessionReportingInterval:(NSInteger) value;

- (FlurrySessionBuilder*) withTVEventCountThreshold:(NSInteger) value;

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

6. FlurryDelegate:

Starting 6.3.0, Flurry analytics can take a delegate to callback on session getting created. A new protocol, FlurryDelegate is added to Flurry Analytics. The delegate object for Flurry needs to implement protocol, FlurryDelegate and need to be set as delegate using the following API call:

```
[Flurry setDelegate:<object that implements FlurryDelegate>];
```

Once delegate is set and the delegate responds to selector `flurrySessionDidCreateWithInfo:`, delegate gets callback on session getting created every time. The info provided in the callback is a NSDictionary object which contains data, “apiKey” and “sessionId”.

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
}
```

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:

- Flurry Pulse - Share your app data with Pulse integrated partners, such as ComScore.
- 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

The following methods are recommended to report additional data and enhance understanding of user behavior in your app. For a list of all APIs support by the Flurry Analytics SDK please see:http://flurrydev.github.io/FlurryiOSSDK6xAPI/interface_flurry.html

Pulse Monitoring

Starting version 6.1.0 Flurry provides means to turn on sending session data to integrated partners. You need to turn on Pulse using the following API call before call to start Session.

```
[Flurry setPulseEnabled:YES];
```

Flurry SDK automatically queries the config end points selected by you and send url's appropriately.

Starting 7.2.0 release, pulse reporting supports application events. Custom application events to be reported can be selected in the Pulse UI. On selecting appropriate partners from Pulse UI, SDK communicates the events triggered to the partners seamlessly. Flurry SDK also maintains log of the partner end-point invocation.

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. 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) .

```
[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.

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.

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 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.

```
[Flurry setPulseEnabled:(BOOL) pulseEnabled];
```

Enable Flurry Pulse. Please visit https://developer.yahoo.com/flurry-pulse/ for more details.

Crash Reporting

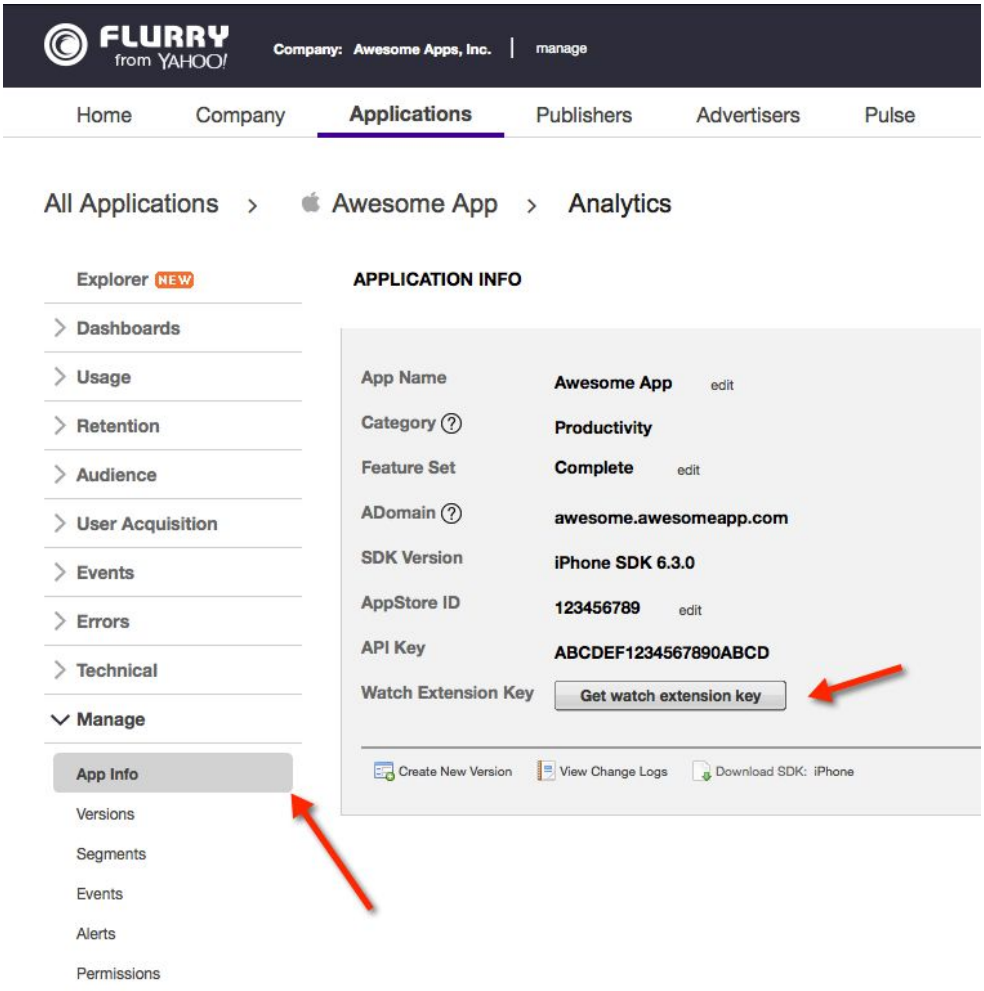
```
[Flurry setCrashReportingEnabled:(BOOL)crashReportingEnabled];
```

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 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.

4. Watch Analytics

Support for Apple Watch extensions require the use of a specific Watch Extension Key. A Watch Extension Key can be acquired by clicking the “Get watch extension key” button on the Manage > App Info page for the parent application. Flurry Analytics on Watch works with watch extensions of your main app. watchOS 1 and watchOS 2 are supported.



Integrating Analytics with a WatchKit extension requires setup in the watch extension’s Info.plist.
In the plist, include a key for “Flurry Watch API Key”, which should include the Watch Extension Key created via the process described above.

PLEASE NOTE: Use caution when entering application APIKeys and Watch Extension Keys into your app code. The incorrect placement of keys will result in invalid metrics.

There are also three optional keys:

1. Flurry Watch App Version - the version of the developers app
2. Flurry Watch Duration Threshold - the maximum time a watch session can exist before being sent to the server. Note that the default value for this property is 10 minutes. 5 minutes is the minimum and 60 minutes is the maximum.
3. Flurry Watch Event Threshold - the maximum number of events that will be recorded in a watch session before being sent to the server. Note that the default value for this is 10 events. There is also a minimum threshold of 5 events and maximum of 50 events that are limits on this property.
4. Flurry Watch Debug - set this to YES to enable debug logs.

Within the watch extension, include FlurryWatch.h.

This gives access to the watch related log event methods:

- `[FlurryWatch logWatchEvent:(NSString*) eventName];`
- `[FlurryWatch logWatchEvent:(NSString*) eventName withParameters:(NSDictionary*) parameters];`

These methods behave much like their counterparts in Flurry.h, but are intended for use only in a watch extension.

Note: When setting up a watchkit extension, the library and frameworks required for Flurry Analytics must also be included in the Linked Frameworks and Libraries for the watch extension target.
For watchOS 1, these are the required libraries:

- libFlurry.a
- Security.framework
- SystemConfiguration.framework

For watchOS 2, the following are required:

- libFlurryWatch.a
- WatchConnectivity.framework

To integrate with Swift applications:

- Please follow the steps listed under the Integrate with Swift applications subsection in the Introduction section.
- In addition to Flurry.h, also import FlurryWatch.h into the bridge header file.
- Use the analogous swift calls to log events.

Objective - C

```
#import "FlurryWatch.h"

- (IBAction) theButtonPressed:(WKInterfaceButton*) sender
   [FlurryWatch logWatchEvent:@"The Button has been pressed"];

   //your code
}
```

Swift Code :

```
@IBAction func theButtonPressed(sender:WKInterfaceButton) {

   FlurryWatch.logWatchEvent("The Button has been pressed")

   //your code
}
```

Watch Analytics FAQ

Why are the watch properties stored in a plist?

Unlike a regular iOS app, which has a single point of entry (AppDelegate), WatchKit extensions have multiple points of entry. InterfaceControllers, NotificationControllers, GlanceControllers can all be called without going through a common piece of code. To avoid having to repeatedly set these values in every WKInterfaceController, they are set in one place.

What are the duration and event thresholds for?

Watch sessions behave a little differently due to the nature of the WKInterfaceController lifecycles. There is no clear indication of when a user has stopped using the watch. With the flip of a wrist they can start and stop watch interactions. To avoid creating large numbers of miniscule sessions, Flurry keeps sessions alive until they reach particular duration or event based thresholds. Once the thresholds are reached, the session ends and a new session begins on the next event.

Does this version support Watch2.0?

Yes, Starting with 7.5.0 FlurryWatch library is included in the release package to support Analytics on WatchOS 2. This library is also available as a Cocoapod.

5. Revenue Analytics

Setup
If already using Flurry pods

1) Replace the libFlurry.a file with the new one
2) Replace the Flurry.h file with the new one

If not already using Flurry

1) Drop the libFlurry.a into your project
2) Add the Flurry.h file into your project

Revenue Analytics brings in 2 new changes:

1) **StoreKit** framework needs to be linked in
2) Public API to enable these changes have been added. There are two ways of going about it:

A) *Feature Flag for automatic logging IAP events (recommended)*

Session builder option:
The FlurrySessionBuilder.withIAPReportingEnabled API enables automatic tracking of all Apple store transactions and needs to be called while creating the session.

```
(FlurrySessionBuilder*) withIAPReportingEnabled:(BOOL) value
```

After session creation:
The setIAPReportingEnabled API can be called anytime to start automatic tracking of all Apple store transactions if it was not already done during session creation.

```
(void) setIAPReportingEnabled:(BOOL)value
```

B) Explicit IAP Logging API

The logPaymentTransaction API needs to be called before the transaction is finished.

```
(void)logPaymentTransaction:(SKPaymentTransaction*)transaction statusCallback:
(void(^)(FlurryTransactionRecordStatus))statusCallback
```

The logPaymentTransaction API also returns a status back

```
typedef enum {
      FlurryTransactionRecordFailed = 0,
      FlurryTransactionRecorded,
      FlurryTransactionRecordExceeded,
      FlurryTransactionRecodingDisabled
} FlurryTransactionRecordStatus;
```

Integration for explicit IAP event logging

Within iOS, Revenue Analytics transactions are logged with a method in the Flurry SDK that takes the SKPaymentTransaction object as a parameter and from that, parses out all required information. This means that you do not need to provide individual parameters for things such as productId or currency. In addition, this method gets all the necessary information to support Receipt Validation (see below).

Objective - C

```
#pragma mark - Payment Transaction Observer

- (void)paymentQueue:(SKPaymentQueue *)queue
updatedTransactions:(NSArray<SKPaymentTransaction *> *)transactions
{
    for (SKPaymentTransaction *transaction in transactions) {
        switch (transaction.transactionState) {
            case SKPaymentTransactionStatePurchased:
                NSLog(@"Payment went through successfully!");

                [Flurry logPaymentTransaction:transaction
statusCallback:^(FlurryTransactionRecordStatus status) {

                }];

                [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
                break;
            case SKPaymentTransactionStateFailed:
                NSLog(@"Payment was a failure.");
                [Flurry logPaymentTransaction:transaction
statusCallback:^(FlurryTransactionRecordStatus status) {
```

```
                }];
                [[SKPaymentQueue defaultQueue] finishTransaction:transaction];
                break;

            default:
                NSLog(@"Not a failure nor a success! => %ld",
transaction.transactionState);
                break;
        }
    }
}
```

Swift Code :

```
func paymentQueue(queue : SKPaymentQueue, transactions : [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .Purchased:
            print("Payment went through successfully!")
            Flurry.logPaymentTransaction(transaction) {status in print("\(status)")
}
            SKPaymentQueue.defaultQueue().finishTransaction(transaction)
        case .Failed:
            print("Payment was a failure.")
            Flurry.logPaymentTransaction(transaction) { status in
print("\(status)")}
            SKPaymentQueue.defaultQueue().finishTransaction(transaction)
        default:
            print("Not a failure nor a success! => \(transaction.transactionState)")
            break
        }
    }
}
```

6. 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 SDK 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 SDK 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 SDK 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.

What iOS Versions does the iOS SDK support?

This Flurry iOS SDK is a fat binary that includes slices for armv7, arm64, i386 and x86_64. Support is provided for iOS 6.0 to iOS 9.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 the Agent support Bitcode?

Yes, Starting iOS SDK version 7.1.0, Flurry Analytics and Advertising SDKs are bit code compatible.

Is Flurry SDK compliant with Apple's ATS requirements?

Yes, all iOS Flurry SDKs are fully compliant with ATS since version 7.2.0 released September 2015. Flurry does not require any special exceptions to be added to your apps.

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

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