Flurry Advertising

iOS SDK Instructions

SDK version 6.7.0
Updated: 07/24/2015

Welcome to Flurry Advertising!

This README contains:

1. Introduction

Flurry provides a flexible ad serving solution to easily manage the complete monetization of your mobile applications. Rooted in Flurry Analytics, integration with Yahoo Gemini, Flurry's Real-time Bidder (RTB) Marketplace, and various mobile ad networks can easily generate the maximum value from ad inventory for publishers.

With Flurry Ads you will be able to:
1. Define your inventory
2. Traffic ad campaigns
3. Track & optimize your performance

The Flurry Ads SDK is modular and contains only the functionality related to serving advertisements. It is designed to be as easy as possible with a basic setup completed in under 10 minutes.

For ad space setup and more information on Flurry Ads, please visit
https://developer.yahoo.com/flurry/docs/publisher/
Please note, it is **required** that you create ad spaces before retrieving ads. Ad spaces can be created on the Flurry Developer Portal or in the application code. If Ad spaces are created in the code, they will appear in the dev portal designated as "Determined by SDK" in the Ad space setup page.

These instructions assume that you have already integrated Flurry Analytics into your application. If you

have not done so, please refer to **Analytics-README** to get started.

2. Advertising API

Starting with version 6.0.0, developers can create objects for Banners and Interstitials. Individual objects control life cycle of an Ad. Support for legacy API using static methods is now deprecated. Developers are encouraged to use Objects based API for serving Ads. The examples in this README will only contain references to the new Object-based API. For information on using the legacy deprecated Static API, please see https://developer.yahoo.com/flurry/docs/publisher/code/iosv5/.

3. Basic 10 Minute Integration

Follow these steps to quickly integrate Flurry Ads into your app:

1. Add the required frameworks**.** Flurry Ads will throw a linking error without the framework, and ads will not display**.**
   - AdSupport.framework
   - CoreGraphics.framework
   - Foundation.framework
   - MediaPlayer.framework
   - StoreKit.framework
   - UIKit.framework
   - AVFoundation.framework
   - CoreMedia.framework
   - libz.dylib
2. In the finder, drag FlurryAds/ into project's file folder.
3. Add it to your project in Xcode: Project > Add to project > FlurryAds - Choose 'Recursively create groups for any added folders'
4. Please make sure `startSession` is called in your app's AppDelegate child class.

The final integration will look as follows:

Objective - C

```
#import "Flurry.h"
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
      [Flurry startSession:@"YOUR_API_KEY"];
      //your code
}

/**
* You can connect Ads in any existing placement in your app, but for
*demonstration purposes we present integration within your ViewController
*/
```

```
#import "FlurryAdBanner.h"
#import "FlurryAdBannerDelegate.h"
#import "FlurryAdInterstitial.h"
#import "FlurryAdInterstitialDelegate.h"

- (void)viewDidAppear:(BOOL)animated {
      [super viewDidAppear:animated];
      /**
      *     We will show banner and interstitial integrations here.
      *
      */

      // 1. Fetch and display banner ads
      FlurryAdBanner *adBanner = [[[FlurryAdBanner alloc]
initWithSpace:@"BANNER_MAIN_VC"] autorelease];
      adBanner.adDelegate = self;
      [adBanner fetchAndDisplayAdInView:self.view
viewControllerForPresentation:self];

      // 2. Fetch fullscreen ads for later display
     FlurryAdInterstitial *adInterstitial = [[[FlurryAdInterstitial alloc]
initWithSpace:@"INTERSTITIAL_MAIN_VC"] autorelease];
      adInterstitial.adDelegate = self;
      adInterstitial.targeting = [FlurryAdTargeting targeting];
      [adInterstitial fetchAd];
}
```

Swift Code :

```
func application(UIApplication, didFinishLaunchingWithOptions launchOptions:
NSDictionary) -> Bool {
       Flurry.startSession("YOUR_API_KEY")
       // your code
       return true
}

override func viewDidAppear(animated: Bool) {
       super.viewDidAppear(animated)

       // 1. Fetch and display banner ads
       let adBanner = FlurryAdBanner(space: "BANNER_MAIN_VC")
       adBanner.adDelegate = self
       adBanner.fetchAndDisplayAdInView(self.view,
viewControllerForPresentation: self)

       // 2. Fetch fullscreen ads for later display
       let adInterstitial = FlurryAdInterstitial(space:
"INTERSTITIAL_MAIN_VC")
```

```
        adInterstitial.adDelegate = self
        adInterstitial.targeting = FlurryAdTargeting()
        adInterstitial.fetchAd()
}
```

If you are integrating banner ads only, you are done - no further action is required. *[adBanner* `fetchAndDisplayAdInView` *]* will display the banner ads and will keep refreshing them until the adBanner object is released.

If you are integrating Interstitial ads, read on.

Objective C:

```
// Most often there is a point in your app to invoke a takeover (e.g. -
button is pressed, level is completed, etc). Here we will be mocking this
existence of a button method that shows full screen ads
- (IBAction) showFullScreenAdClickedButton:(id)sender {
// Check if ad is ready. If so, display the ad
if (adInterstitial != nil && [adInterstitial ready]) {
      [adInterstitial presentWithViewController:self]
      } else {
          // fetch an ad
          [adInterstitial fetchAd];
      }
}

/*
*  It is recommended to pause app activities when an interstitial is shown.
*  Listen to adInterstitialWillPresent delegate.
*/
- (void) adInterstitialWillPresent:(FlurryAdInterstitial *)flurryAd
{
      // Pause app state here
}

/*
*  Resume app state when the interstitial is dismissed.
*/
- (void) adInterstitialDidDismiss:(FlurryAdInterstitial *)flurryAd
{
// Resume app state here
}
```

Swift:

```
// Most often there is a point in your app to invoke a takeover (e.g. -
button is pressed, level is completed, etc). Here we will be mocking this
existence of a button method that shows full screen ads
    @IBAction func showFullScreenAdClickedButton(sender: AnyObject) {
        if adInterstitial != nil && adInterstitial.ready {
            adInterstitial.presentWithViewController(self)
        } else {
            adInterstitial.fetchAd()
        }
    }

    /*
    *  It is recommended to pause app activities when an interstitial is
shown.
    *  Listen to adInterstitialWillPresent delegate.
    */
    func adInterstitialWillPresent(interstitialAd: FlurryAdInterstitial!) {
        // Pause app state here
    }

    /*
    *  Resume app state when the interstitial is dismissed.
    */
    func adInterstitialWillDismiss(interstitialAd: FlurryAdInterstitial!) {
        // Resume app state here
    }
```

For more details on the Advertising API please refer to
https://developer.yahoo.com/flurry/docs/publisher/code/ios/
Also note that the SDK package downloaded from the developer site contains API documentation.

4. Requesting Native Ads

To request native ads in your code, first start session using Flurry API. Note: please see **Analytics-README** for details on [Flurry startSession:]. Native ad object should be created using FlurryAdNative class with initialization method `initWithSpace:`.

4.1 Set up the ad

Set the adDelegate property on the native ad object to the object that will implement the FlurryAdNativeDelegate protocol. Set the viewControllerForPresentation to the UIViewcontroller that will be designated to display the native ads.

4.2 Asynchronously fetching an ad

Call the fetchAd routine on FlurryAdNative to fetch an ad asynchronously. This enables you to pre-load ads before they are actually displayed. Use the following call to fetch an ad

```
- (void)fetchAd;
```

Once you have made the request for an ad to be fetched, there are two ways of proceeding to display the ad. The first is to check if the ad is ready at various times, and then display the ad once it is ready. The other way is to implement the `<FlurryAdNativeDelegate>` which will be notified with a call to `adNativeDidFetchAd` when the ad is ready. It is recommended to implement the `adNative:adError:errorDescription:` method to get notified of a failure to fetch ads.

4.3 Checking if an ad is ready

After an ad is fetched, you can explicitly check if the ad is ready to be displayed from a property of `FlurryAdNative` object, `ready`.

```
@property (nonatomic, readonly) BOOL ready;
```

The value of this property is YES or NO based on availability of the ad.

4.4 Checking if an ad has expired

After an ad is fetched, you can explicitly check if the ad is expired from a property of `FlurryAdNative` object, `expired`.

```
@property (nonatomic, readonly) BOOL expired;
```

The value of this property is YES or NO based on expiration state of the ad.

4.5 Using the Native Ad assets

When the ad is ready the native ads asset list will be available and can be accessed using the assetList property.

```
@property (nonatomic, readonly) assetList;
```

This property returns an array of FlurryAdNativeAsset objects. Each object has details about the asset such as:

```
@property (nonatomic, retain, readonly) NSString *name;
@property (nonatomic, readonly) kAssetType type;
@property (nonatomic, retain, readonly) NSString *value;
@property (nonatomic, assign, readonly) int width;
@property (nonatomic, assign, readonly) int height;
```

The `type` property of an asset is an enum value of type `kAssetType`. This indicates whether the asset is video, image or text. The property, `value` contains the text or url for the asset. If an image asset is precached, the url will be a file url pointing to cached asset on the device.

These assets can then be used to populate a UIView within your application and can be used to display the native ad.

4.6 Tracking Ad

After an ad is placed in appropriate location in your app, the UIView object used to create the ad needs to be set as tracking view for the native ad object. Once the property, `trackingView` of `FlurryAdNative` object is set, it is used to track the viewability of the native ad automatically. The viewability criteria for native ads is based on IAB guidelines (50% of ad view visible for more than 1 second). Once the ad view met this criteria, impression urls are fired to log views.

In addition to impression tracking, Flurry SDK provides automatic click tracking of ad when the ad view is set as `trackingView` of `FlurryAdNative` object.

Note: It is very important to set the ad view you created to be tracking view of native ad object for proper monetization in the app.
Example usage of native ad in a custom table view cell as follows:

```
@interface AdStreamCell : UITableViewCell

@property (nonatomic, retain) FlurryAdNative* ad;

@end

@implementation AdStreamCell

- (void) setupAdCellForNativeAd:(FlurryAdNative*) nativeAd
{
    [self.ad removeTrackingView];
    self.ad = nativeAd;
    self.ad.trackingView = self;
}
```

The basic steps for integration of native ads were listed above. For more details including code

samples on native ad integrations please refer to the Flurry support site:
https://developer.yahoo.com/flurry/docs/publisher/code/ios/#native-display-integration-code

5. Video Ads

Flurry supports Video ad on the Interstitial ad format. Publishers have the option of configuring length of video play allowed, if the video can be skipped, and if a reward should be granted for watching a video. By default, all video ads on the Flurry network are pre-cached for enhanced user experience and completion rates. To set up your interstitial ad space for video see
https://developer.yahoo.com/flurry/docs/howtos/videoadspace/

5.1 Videos in Native Ads

Starting with 6.7.0, Flurry supports video ads in Native ad format. The structure of video ad is identical to image based native ad except for videoViewContainer property which should be non-nil.

A non nil UIView object need to be provided for videoViewContainer in addition to trackingView. The ad video will be prepared and rendered in the videoViewContainer by SDK. Please note video asset is not provided in the asset list of native ad object.

Video view will be expanded to full screen when end user taps in video area. A tap on other regions of the ad lead to display appropriate pages/views. Please note that all tap gestures are reset on the tracking view. End users can pause/restart/mute the video in full screen display.

Note: It is very important to set the videoViewContainer property of native ad to be able to display and monetize the video.

Videos inside the ads can be set to auto play in Flurry dev portal. Videos set to “autoplay” are played as soon as the viewability criteria (50% of tracking view in viewable area for 1 continuous second) is met. Auto playing videos are paused as soon the ad view is out of viewable region.

5.1.1 Checking if an ad is video ad

FlurryAdNative object API provides a method, “isVideoAd” to check whether an ad fetched is video based ad or not. Call this method to find if you need to provide videoviewcontainer.

```
@interface AdStreamCell : UITableViewCell

@property (nonatomic, retain) FlurryAdNative* ad;
@property (weak, nonatomic) IBOutlet UIView *cardVideoViewContainer;
@end

@implementation AdStreamCell

- (void) setupAdCellForNativeAd:(FlurryAdNative*) nativeAd
{
    // Please make sure removeTrackingView is called on the FlurryAdNative
```

```
object being replaced.
    [self.ad removeTrackingView];

    self.ad = nativeAd;
    self.ad.trackingView = self;
    if([self.ad isVideoAd])
    {
       self.ad.videoViewContainer = nil;
       self.ad.videoViewContainer =  self.cardVideoViewContainer;
    }
}
```

6. Ad Targeting (Optional)

Flurry ADS API supports ad targeting using an object of `FlurryAdTargeting`. Both `FlurryAdBanner` and `FlurryAdInterstitial` have a property `targeting`. Publishers can control the ads using various properties of the `FlurryAdTargeting` object.

FlurryAdTargeting is a container for various properties to target an ad space effectively such as:

```
@property (nonatomic, retain) CLLocation* location;
@property (nonatomic, retain) NSDictionary *userCookies;
@property (nonatomic, retain) NSDictionary *keywords;
@property (nonatomic, assign) BOOL testAdsEnabled;
```

6.1 `location`: If your app collects location information you should pass that to Flurry. This will result in better targeting and higher spend from advertisers. The user device location information can be attached to FlurryAdTargeting object if the app has permission to do so. Once app obtains permission, FlurryAd objects send the location information to target effectively.

6.2 `userCookies`: UserCookies allow the developer to specify information on a user executing an ad action. On ad click UserCookie key/value is transmitted to the Flurry servers. The UserCookie key/value pairs will be transmitted back to the developer via the app callback if one is set. This is useful for rewarded inventory, to identify which of your users should be rewarded when a reward callback is sent.

6.3 `keywords`: Keywords allow the developer to specify information on a user executing an ad action for the purposes of targeting. There is one keywords object that is transmitted to the Flurry servers on each ad request. If corresponding keywords are matched on the ad server, a subset of targeted ads will be delivered. This allows partners to supply information they track internally, which is not available to Flurry's targeting system.

6.4 `testAdsEnabled`: Publishers can use this field to fetch test ads from flurry server during app development. Test ads won't monetize and are provided for integration test only. Please make sure to set to false before the app is launched.

7. Implementing Ad Delegate (Optional)

7.1 FlurryAdBannerDelegate

To be notified of events during life cycle of FlurryAdBanner, the calling object need to implement FlurryAdBannerDelegate protocol. And the calling object need to set itself as the delegate to FlurryAdBanner object.
Please refer to http://flurrydev.github.io/FlurryiOSSDK6xAPI/protocol_flurry_ad_banner_delegate-p.html for for additional details on this delegate's callback methods.

7.2 FlurryAdInterstitialDelegate

To be notified of events during life cycle of FlurryAdInterstitial, the calling object need to implement FlurryAdInterstitialDelegate protocol. And the calling object need to set itself as the delegate to FlurryAdInterstitial object.
Please refer to http://flurrydev.github.io/FlurryiOSSDK6xAPI/protocol_flurry_ad_interstitial_delegate-p.html for additional details on this delegate's callback methods.

7.3 FlurryAdNativeDelegate

To be notified of events during life cycle of FlurryAdNative, the calling object need to implement FlurryAdNativeDelegate protocol. And the calling object need to set itself as the delegate to FlurryAdNative object.

Please refer to http://flurrydev.github.io/FlurryiOSSDK6xAPI/protocol_flurry_ad_native_delegate-p.html for additional details on this delegate's callback methods.

8. Enabling Ad Network Mediation (Optional)

Once your Ad Spaces are configured, you have the option of selecting 3rd party ad networks to serve ads into your Ad Spaces. You can change which ad networks serve ads at any time on the Flurry website, but in order to enable them you need to add the ad network SDKs into your application and configure them. The following Ad Networks are currently supported:

- iAd
- Admob - Framework version 7.0.0
- Millennial - Framework version 5.4.1
- InMobi - SDK version 4.5.1

Please refer to https://developer.yahoo.com/flurry/docs/publisher/gettingstarted/targeting/ for more information regarding setting up Ad Network Mediation.

9. F.A.Q.

How do you set a device as a test device?

The Flurry SDK can be configured to receive test ads. This can be used to test and visually verify integrations. Test ads do not generate revenue and therefore MUST be disabled before submitting to the AppStore. Please see https://developer.yahoo.com/flurry/docs/publisher/code/ios/ for more details on this feature.