FLURRY

Flurry Advertising

iOS SDK Instructions

SDK version 6.3.0
Updated: 3/26/2015

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 Flurry's Real-time Bidder (RTB) Marketplace, Flurry's own network, 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. Objects based API

Major advancement in Flurry API with this newest release is that the Flurry Ads SDK API is now Objects based. Starting from 6.0.0 version, developers can create objects for Banners and Interstitials. Individual objects control life cycle of an Ad. Support for legacy API using static methods is still available but will be deprecated soon. Developers are encouraged to use Objects based API for serving Ads. Quick integration steps are explained in each section of this document.

3. Basic 10 Minute Integration

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

3.1 Using Objects based API:

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
   - 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: "MoPubTakeover")
        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 Integrating Native Ads please refer to section 6.

3.2 Using legacy Static API:

1. **The Ad Support framework is required. Flurry Ads will throw a linking error without the framework, and ads will not display.**
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. In your source code, import FlurryAds and initialize FlurryAds with the initialize call sometime after

`startSession`.

The parameter is as follows:

· UIViewController rvc - The root view controller of your application window

The final integration will look as follows:

Objective C:

```
#import "Flurry.h"
#import "FlurryAds.h"

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
      [Flurry startSession:@"YOUR_API_KEY"];
      [FlurryAds initialize:window.rootViewController];
                        // pointer to your rootViewController
      //your code
}

/**
*      You can connect Ads in any existing placement in your app, but for
demonstration purposes we present integration within your ViewController
*/
#import "FlurryAdDelegate.h"
- (void)viewDidAppear:(BOOL)animated {
      [super viewDidAppear:animated];
      /**
      *      We will show banner and interstitial integrations here.
      *
      */
      // Register this object as a delegate for ad callbacks
      [FlurryAds setAdDelegate:self];

      // 1. Fetch and display banner ads
      [FlurryAds fetchAndDisplayAdForSpace:@"BANNER_MAIN_VC" view:self.view
viewController:self size:BANNER_BOTTOM];

      // 2. Fetch fullscreen ads for later display
      [FlurryAds fetchAdForSpace:@"INTERSTITIAL_MAIN_VC"
frame:self.view.frame size:FULLSCREEN];

}
```

Important Note: Do not set ad delegate to nil and do not call remove ad in the viewWillDisappear or viewDidDisappear method of the presenting view controller that is passed into fetchAndDisplayAdForSpace: and displayAdForSpace:onView:viewControllerForPresentation routines.

When a full screen ad is shown on the view controller passed into these routines then the viewWillDisappear and viewDidDisappear methods of the view controller will get called. Similarly, clicking on a banner could show a full screen ad. Please remove Ad when the view controller is getting deallocated or navigated away by any controllers in the app. This is required for clean up and to avoid persistent banner refreshes upon view controller navigated away by user.

Swift:

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

        FlurryAds.setAdDelegate(self)

        // 1. Fetch and display banner ads
        FlurryAds.fetchAndDisplayAdForSpace("BANNER_MAIN_VC", view:self.view,
viewController:self, size:BANNER_BOTTOM)

        // 2. Fetch fullscreen ads for later display
        FlurryAds.fetchAdForSpace("INTERSTITIAL_MAIN_VC",
frame:self.view.frame,
      size:FULLSCREEN);

}
```

If you are integrating banner ads only, you are done - no further action is required. *[FlurryAds fetchAndDisplayAdForSpace]* will display the banner ads and will keep refreshing them until you call *[FlurryAds removeAdFromSpace]*

If you are integrating Takeover (Interstitial) ads read on.

```
// 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 ([FlurryAds adReadyForSpace: :@"INTERSTITAL_MAIN_VC"]) {
        [FlurryAds displayAdForSpace: :@"INTERSTITAL_MAIN_VC"
onView:self.view viewControllerForPresentation:self];
    } else {
        // fetch an ad
        [FlurryAds fetchAdForSpace:@"INTERSTITAL_MAIN_VC"
frame:self.view.frame size:FULLSCREEN];
    }
}

/*
 *  It is recommended to pause app activities when an interstitial is shown.
 *  Listen to should display delegate.
 */
- (BOOL) spaceShouldDisplay:(NSString*)adSpace interstitial:(BOOL)
    interstitial {
   if (interstitial) {
     // Pause app state here
   }
   // Continue ad display
    return YES;
}

/*
 *  Resume app state when the interstitial is dismissed.
 */
- (void)spaceDidDismiss:(NSString *)adSpace interstitial:(BOOL)interstitial {
    if (interstitial) {
     // Resume app state here
    }
}
```

4. Requesting Banner Ads

In order to specify the format and size, you should log into the Flurry Developer Portal at http://dev.flurry.com. Under the Publishers tab, select Inventory / Ad Spaces to specify the configuration for the ad space name you have chosen.

4.1 Using Object API

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

The method, `fetchAndDisplayAdInView: viewControllerForPresentation:` should be called to start fetching and displaying banner ads that get refreshed automatically.

If you want to fetch ads first and display later, fetch the ads by calling `fetchAdForFrame:` method on the banner object. When you are ready to display banner, call `displayAdInView: viewControllerForPresentation:` to display and refresh automatically from that point on.

One of the banner object's properties, `ready` can be used to find if the banner object is ready to display. And, an object which implements `FlurryADBannerDelegate` protocol need to register itself as a delegate to the banner object for getting callbacks during banner life cycle.

FlurryAdBanner methods:

```
- (id) initWithSpace:(NSString *)space;
```

Parameters:

- **NSString space** - Name of the adSpace that you defined on the Flurry website.

```
- (void) fetchAndDisplayAdInView:(UIView *)view
viewControllerForPresentation:(UIViewController *)viewController;
```

Parameters:

- **UIView viewContainer** - UIView that you want the ad to reside in.
- **UIViewController viewController** - UIViewController that will be used for presenting fullscreen ads. The fullscreen will be presented modally on this view controller.
- **FlurryAdSize size** - The default size for your adSpace. This is overwritten by any value set on the server.

```
- (void) fetchAdForFrame:(CGRect)frame;
```

Parameters:

- **CGFrame frame:** CGFrame for size of the view to be created to house banner ad.
  e.g. a CGFrame of valid banner size: 320 x 50

```
- (void) displayAdInView:(UIView *)view
viewControllerForPresentation:(UIViewController *)viewController;
```

Parameters:

- **UIView view** - UIView that you want the ad to reside in.
- **UIViewController viewController** - UIViewController that will be used for presenting ads. The full screen display of ad when clicked will be presented modally on this view controller.

FlurryAdBanner object properties:

To check if the banner is ready to display:

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

Delegate to get callbacks from banner object during ad life cycle:

```
@property (nonatomic, assign) id<FlurryAdBannerDelegate> adDelegate;
```

To target user based on location or keywords

```
@property (nonatomic, retain) FlurryAdTargeting* targeting;
```

4.2 Using Static API

To request ads in your code, call `[FlurryAds fetchAndDisplayAdForSpace: view: viewController:size]` after the session starts. Note: please see **Analytics-README** for details on [Flurry startSession:]

```
+ (void)fetchAndDisplayAdForSpace:(NSString*)space view:(UIView*)viewContainer
viewController:(UIViewController*) viewControllerForPresentation
size:(FlurryAdSize)size;
```

The parameters are as follows:

- **NSString space** - Name of the adSpace that you defined on the Flurry website.
- **UIView viewContainer** - UIView that you want the ad to reside in.
- **UIViewController viewController** - UIViewController that will be used for presenting fullscreen ads. The fullscreen will be presented modally on this view controller.
- **FlurryAdSize size** - The default size for your adSpace. This is overwritten by any value set on the server.

Although this ad call can be used for all ad formats, note that using this method the ad will be displayed as soon as it is ready. We recommend this method be used for Banner and Custom ad sizes.

For Takeover ads, we recommend asynchronous set that separates fetch from display to allows for finer control over when the ad will be displayed.

5. Requesting Takeover Ads

5.1 Using Object API

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

5.1.1 Asynchronously fetching an ad

FlurryAdInterstitial provides a method to fetch and store an ad asynchronously before it is displayed. 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 `<FlurryAdIntertitialDelegate>` which will be notified with a call to `adInterstitialDidFetchAd` when the ad is ready. You can then call `presentWithViewControler:` at your convenience. For details on `FlurryAdInterstitialDelegate`, see section 8 below, named Implementing Ad Delegates.

5.1.2 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 FlurryAdInterstitial object, `ready`.

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

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

5.1.3 Displaying the fetched ad

Once the ad is fetched, it can be displayed with the following call. Make sure to use `ready` property to ensure that an interstitial ad can be displayed.

```
- (void) presentWithViewControler: (UIViewController *)rvc;
```

Parameters:

- **UIViewController viewController** - UIViewController that will be used for presenting fullscreen ads. The fullscreen will be presented modally on this view controller.

5.2 Using Static API

5.2.1 Asynchronously fetching an ad

Flurry provides a method to fetch and store an ad asynchronously before it is displayed. This enables you to pre-load ads before they are actually displayed. Use the following call to fetch an ad

```
+ (void)fetchAdForSpace:(NSString*)space frame:(CGRect)frame
size:(FlurryAdSize)size;
```

The parameters are as follows:

- **NSString space** - Name of the adSpace that you defined on the Flurry website.
- **CGRect frame** - The frame of the UIView that you want the ad to reside in.
- **FlurryAdSize size** - The default size for your adSpace. This is overwritten by any value set on the server.

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 (see calls for `adReadyForSpace` and `displayAdForSpace` below). The other way is to implement the `<FlurryAdDelegate>` which will be notified with a call to `spaceDidReceiveAd` when the ad is ready. You can then call `displayAdForSpace` at your convenience. For details on `FlurryAdDelegate`, see section 7: Implementing Ad Delegate.

5.2.2 Checking if an ad is ready

After an ad is fetched, you can explicitly check if the ad is ready to be displayed.

```
+ (BOOL)adReadyForSpace:(NSString*)space
```

This call will check if an ad is ready. If an ad is ready to be displayed then this will return true, or false if the ad is not ready.

- **NSString space** - Name of the adSpace that you defined on the Flurry website

5.2.3 Displaying the fetched ad

Once the ad is fetched, it can be displayed with the following call. Make sure to use `adReadyForSpace` above to ensure that an ad can be displayed.

```
+ (void)displayAdForSpace:(NSString*)space onView:(UIView*)viewContainer
viewControllerForPresentation:self;
```

The parameters are as follows:

- **NSString space** - Name of the adSpace that you defined on the Flurry website.
- **UIView viewContainer** - UIView that you want the ad to reside in.
- **UIViewController viewController** - UIViewController that will be used for presenting fullscreen ads. The fullscreen will be presented modally on this view controller.

5.2.4 RTB takeover ads

For RTB Interstitial integration instructions please see further instructions:
https://developer.yahoo.com/flurry/docs/faq/faqpublisher/ios/

6. 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:`.

6.1 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.

6.1 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.

6.2 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.

6.3 Checking if an ad is 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.

6.4 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 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.

For more details on native ads please refer to the Flurry support site:
https://developer.yahoo.com/flurry/docs/publisher/gettingstarted/nativeads/ios/

6.5 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;
}
```

7. Video Ads

Starting with version 5.0.0, FlurryAds SDK supports precaching of the video ads for enhanced user experience and better completion rate. Precaching is done for video ads served by Flurry marketplace and by Flurry's own network.

Since version 5.0.0, Flurry SDK supports displaying VAST Linear (including VAST Wrapper) ads from Flurry marketplace.

Above features are supported automatically without any additional integration.

8. Ad Targeting (Optional)

8.1 Using Object API

Starting with version 6.0.0 Flurry SDK's Object 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;
```

8.1.1 `location`: 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.

8.1.2 `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.

8.1.3 `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.

8.1.4 `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.

8.2 Using Static API

8.2.1 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.

Add a call to specify keywords to be used when targeting ads:

```
[FlurryAds setKeywordsForTargeting:(NSDictionary *) keywords];
```

To clear the set keywords call:

```
[FlurryAds clearKeywords];
```

8.2.2 User Cookies

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.

Add a call to identify any user specific information you want associated with the ad request:

```
[FlurryAds setUserCookies:(NSDictionary *) userCookies];
```

To remove any user cookies call:

```
[FlurryAds clearUserCookies];
```

9. Removing an ad

Flurry manages the lifecycle of the ads it displays, however, you can exercise finer control over display by choosing when to add and remove the ads from your app.

It is recommended to not set ad delegate to nil or not call remove ad in the viewWillDisappear or viewDidDisappear method of the presenting view controller that is passed into `fetchAndDisplayAdForSpace:` and `displayAdForSpace:onView:viewControllerForPresentation` routines. When a full screen ad is shown on the view controller passed into these routines then the viewWillDisappear and viewDidDisappear methods of the view controller will get called. Also note, clicking on a banner could show a full screen ad. Please remove ad when the view controller is getting removed or navigated away by controllers in the app. This is required for clean up and to avoid persistent banner refreshes upon view controller navigated away by user.

9.1 Using Object API
To remove an ad the corresponding ad object should be released and/or set to nil.

9.2 Using Static API
To remove an ad just call
```
+ (void)removeAdFromSpace:(NSString*)space;
```

The parameter is as follows:
- **NSString space** - Name of the adSpace that you defined on the Flurry website

10. Request Configuration (Optional)

There are a number of configuration parameters that you can use to modify the behavior of your ad spaces.

Option 1. Enable Test Ads
For testing purposes Flurry SDK ads Add a call to receive test ads from the flurry server to ensure proper implementation. Test ads do not generate revenue and therefore MUST be disabled before submitting to the AppStore:

Using Object API:
Targeting is implemented using `FlurryAdTargeting` objects. Users can create this class and set `testsEnabled` property of the object to YES.

Using Static API:
```
[FlurryAds enableTestAds:(BOOL)enable];
```

Option 2. Set Location

Using Object API:
Location target for ads is set on targeting property (FlurryAdTargeting) of ad objects (both FlurryAdBanner and FlurryAdInterstitial). Once app gets permission from user to get location notifications, the location property of targeting object needs to be set.

```
@property (nonatomic, retain) CLLocation* location;
```

Using Static API:
Add a call to set the location (lat,long) that you want associated with the ad request, to be used with geographical targeting. Passing lat,long data enables better monetization of your ad inventory.

```
[Flurry setLatitude:(double)latitude longitude:(double)longitude
horizontalAccuracy:(float)horizontalAccuracy
verticalAccuracy:(float)verticalAccuracy];
```

Option 3. User Cookies
Please refer above section 7, Ad Targeting for info on this.

Option 4. Keyword Targeting
Please refer above section 7, Ad Targeting for info on this.

11. Implementing Ad Delegate (Optional)

11.1 Object API

11.1.1 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.

Callback methods:

- - (void) adBanner:(FlurryAdBanner*) bannerAd adError:(FlurryAdError) adError errorDescription:(NSError*) errorDescription;
    - Informational callback invoked when there is an ad error
- - (void) adBannerDidFetchAd:(FlurryAdBanner*)bannerAd;
    - Invoked when an ad is received for the specified bannerAd object.
- - (void) adBannerDidRender:(FlurryAdBanner*)bannerAd;
    - This method informs the user an ad was retrieved, and successful in displaying to the user.
- - (void) adBannerWillPresentFullscreen:(FlurryAdBanner*)bannerAd;
    - Invoked when the specified banner ad object is about to present a full screen.
- - (void) adBannerWillLeaveApplication:(FlurryAdBanner*)bannerAd;
    - Invoked when the ad has been selected that will take the user out of the app.
- - (void) adBannerWillDismissFullscreen:(FlurryAdBanner*)bannerAd;
    - Invoked when a fullscreen associated with the specified ad will be removed.
- - (void) adBannerDidDismissFullscreen:(FlurryAdBanner*)bannerAd;
    - Invoked when a fullscreen associated with the specified ad has been removed.
- - (void) adBannerDidReceiveClick:(FlurryAdBanner*)bannerAd;

    - This method informs the app that an ad has been clicked. This should not be used to adjust state of an app. It is only intended for informational purposes.
- - (void) adBannerVideoDidFinish:(FlurryAdBanner*)bannerAd;
    - This method informs the app that a video associated with this ad has finished playing. This method is invoked for Rewarded ad spaces only (this includes the following ad mix options: Rewarded, Let Flurry Decide and Client-side Rewarded). The ad space configured as Standard ad mix will not invoke this delegate.

11.1.2 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.

Callback methods:

- - (void) adInterstitialDidFetchAd:(FlurryAdInterstitial*)interstitialAd;
    - Invoked when an ad is received for the specified interstitialAd object.
- - (void) adInterstitial:(FlurryAdInterstitial*) interstitialAd adError:(FlurryAdError) adError errorDescription:(NSError*) errorDescription;
    - Informational callback invoked when there is an ad error of type adError (FlurryAdError)
- - (void) adInterstitialDidRender:(FlurryAdInterstitial*)interstitialAd;
    - This method informs the user an ad was retrieved, and successful in displaying to the user.
- - (void) adInterstitialWillPresent:(FlurryAdInterstitial*)interstitialAd;
    - Invoked when a fullscreen associated with the specified ad will present on the screen.
- - (void) adInterstitialWillLeaveApplication:(FlurryAdInterstitial*)interstitialAd;
    - Invoked when the ad has been selected that will take the user out of the app.
- - (void) adInterstitialWillDismiss:(FlurryAdInterstitial*)interstitialAd;
    - Invoked when a fullscreen associated with the specified ad will be removed.
- - (void) adInterstitialDidDismiss:(FlurryAdInterstitial*)interstitialAd;
    - Invoked when a fullscreen associated with the specified ad has been removed.
- - (void) adInterstitialDidReceiveClick:(FlurryAdInterstitial*)interstitialAd;
    - Informational callback invoked when an ad is clicked for the specified interstitialAd object. This should not be used to adjust state of an app. It is only intended for informational purposes.
- - (void) adInterstitialVideoDidFinish:(FlurryAdInterstitial*)interstitialAd;
    - This method informs the app that a video associated with this ad has finished playing. This method is invoked for Rewarded ad spaces only (this includes the following ad mix options: Rewarded, Let Flurry Decide and Client-side Rewarded). The ad space configured as Standard ad mix will not invoke this delegate.

11.1.3 FlurryAdNativeDelegate

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.

Callback methods:

- - (void) adNativeDidFetchAd:(FlurryAdNative*) nativeAd;
    - Invoked when an ad has been received and is available for display.
- - (void) adNative:(FlurryAdNative*) nativeAd adError:(FlurryAdError) adError errorDescription:(NSError*) errorDescription;
    - Informational callback invoked when there is an ad error of type adError (FlurryAdError)
- - (void) adNativeWillPresent:(FlurryAdNative*) nativeAd;
    - Invoked when the native ad object that is associated with the full screen that is about to present a fullscreen.
- - (void) adNativeWillLeaveApplication:(FlurryAdNative*) nativeAd;
    - This method informs the app that an ad has been clicked and the user is about to be taken outside the app.
- - (void) adNativeWillDismiss:(FlurryAdNative*) nativeAd;
    - Invoked when a fullscreen associated with the specified ad will be removed.
- - (void) adNativeDidDismiss:(FlurryAdNative*) nativeAd;
    - Invoked when a fullscreen associated with the specified ad has been removed.
- - (void) adNativeDidReceiveClick:(FlurryAdNative*) nativeAd;
    - Informational callback invoked when an ad is clicked for the specified nativeAd object.

11.2 Static API

To be notified of certain events during the full lifecycle of the Ad, you can implement the FlurryAdDelegate and then call the [FlurryAds setDelegate:] method to attach your delegate to the FlurryAds. When you implement the FlurryAdDelegate you will implement the following callback methods:

- (void) spaceDidReceiveAd:(NSString*)adSpace;
    - This method is called when an ad has been received and is available for display on the ad space specified by adSpace.
- (void) spaceDidFailToReceiveAd:(NSString*) adSpace error:(NSError *)error;
    - This method informs the app that an ad has failed to be received for the given adSpace.
- (BOOL) spaceShouldDisplay:(NSString*)adSpace interstitial:(BOOL)interstitial;
    - This method informs the app that an ad is about to be displayed on the adSpace. The parameter interstitial will be YES/NO if the space to display will be an interstitial. You can decide at this point not to show this ad by simply returning NO. Returning YES will allow the ad to be shown.
- (void) spaceDidFailToRender:(NSString *)space error:(NSError *)error;
    - This method informs the user an ad was retrieved, however, was unsuccessful in displaying to the user (could be lost network connectivity for example).
- (void) spaceWillDismiss:(NSString *)adSpace;
    - This method will be called when the user dismisses the current Ad for the provided Ad Space name.
- (void)spaceDidDismiss:(NSString *)adSpace interstitial:(BOOL)interstitial;
    - This method informs the app that an ad has closed. You can use this to resume app states.
- (void) spaceWillLeaveApplication:(NSString *)adSpace
    - This method will be called when the user is leaving the application after following events associated with the current Ad in the provided Ad Space name.
- (void)videoDidFinish:(NSString *)adSpace;

- This method will be called when a video finishes playing to inform the app that a video associated with an ad has finished playing. This method is invoked for Rewarded ad spaces only (this includes the following ad mix options: Rewarded, Let Flurry Decide and Client-side Rewarded). The ad space configured as Standard ad mix will not invoke this delegate. The ad space mix configuration is done on the server side, see further instructions:
- https://developer.yahoo.com/flurry/docs/publisher/gettingstarted/advancedconfiguration/#advanced-ad-space-setup-serving-rewarded-ads and
- https://developer.yahoo.com/flurry/docs/publisher/gettingstarted/in-apprewardedads/

Example usage:

```
@interface MyDelegateClass : NSObject <FlurryAdDelegate> {

// definitions
}

// MyDelegateClass.m

@implementation MyDelegateClass

-(void) init
{
      [super init];
      [FlurryAds setAdDelegate:self];         // Set the delegate
}

- (void) spaceDidReceiveAd:(NSString*)adSpace
{
      // Show the ad if desired
      [FlurryAds displayAdForSpace:[self myAdSpace] onView:[self view]
      viewControllerForPresentation:self];
}

- (void) spaceDidFailToReceiveAd:(NSString*)adSpace error:(NSError *)error
{
      // Handle failure to receive ad
}

- (BOOL) spaceShouldDisplay:(NSString*)adSpace
interstitial:(BOOL)interstitial
{
      // Decide if the Ad should be displayed
      return true;
}
```

```
- (void) spaceDidFailToRender:(NSString *)space error:(NSError *)error
{
      // Handle a failure to render the ad
}

- (void)spaceWillDismiss:(NSString *)adSpace
{
      // Handle the user dismissing the ad
}

- (void)spaceDidDismiss:(NSString *)adSpace
{
      // Handle the closing of the ad
}

- (void)spaceWillLeaveApplication:(NSString *)adSpace
{
      // Handle the user leaving the application
}

- (void)videoDidFinish:(NSString *)adSpace
{
      // Invoked only for rewarded ad spaces
}

@end
```

12. 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

To implement an Ad Network you must perform the following steps:

1. Include the Ad Network iOS SDK with your app and add it to the project. Follow the instructions from the Ad network on how to complete this step.

2. Make fetch ad request and call display Ad API calls appropriately for both banners and Interstitials. Note: For Millennial banners, please use `fetchAndDisplayAdInView` API call instead of fetch and display separately for best results.
3. Implement the appropriate delegate methods in FlurryAdDelegate

Here is the example of implementing the Admob SDK into FlurryAds:

```
@interface MyDelegateClass : NSObject <FlurryAdDelegate> {

 // definitions
}

// MyDelegateClass.m

@implementation MyDelegateClass

      - (NSString *) flurryAdsAdMobPublisherID
      {
            // Your AdMob Publisher ID, found from here:
                  http://www.admob.com/my_sites/ (click manage settings)
            return @"<your id>";
      }

      - (BOOL) flurryAdsTestMode
      {
            // If testing, return yes
            return YES;
      }

@end
```