Appsfire SDK

v2.0 *(Updated on November 21st 2013)*

I. Basics

A. Requirements

This document covers the iOS version of the Appsfire SDK. If you are looking for the Android version, please visit your dashboard.

The Appsfire SDK runs on iOS 5.0 and versions above. It compiles against the armv7 and armv7s architectures. Certain features require that your app have a normalized version number (e.g. 1, 2.0, 2.1.2, 4.1.4.2) and that this version number be properly declared in your main plist. See FAQ.

B. What's New

Version 2.0

- Advertising SDK
- New flat design
- And much more!

Version 1.1.5

- Metrics gathering for Push Text Alerts
- Ability to hide “Send Feedback” button

Version 1.1.4

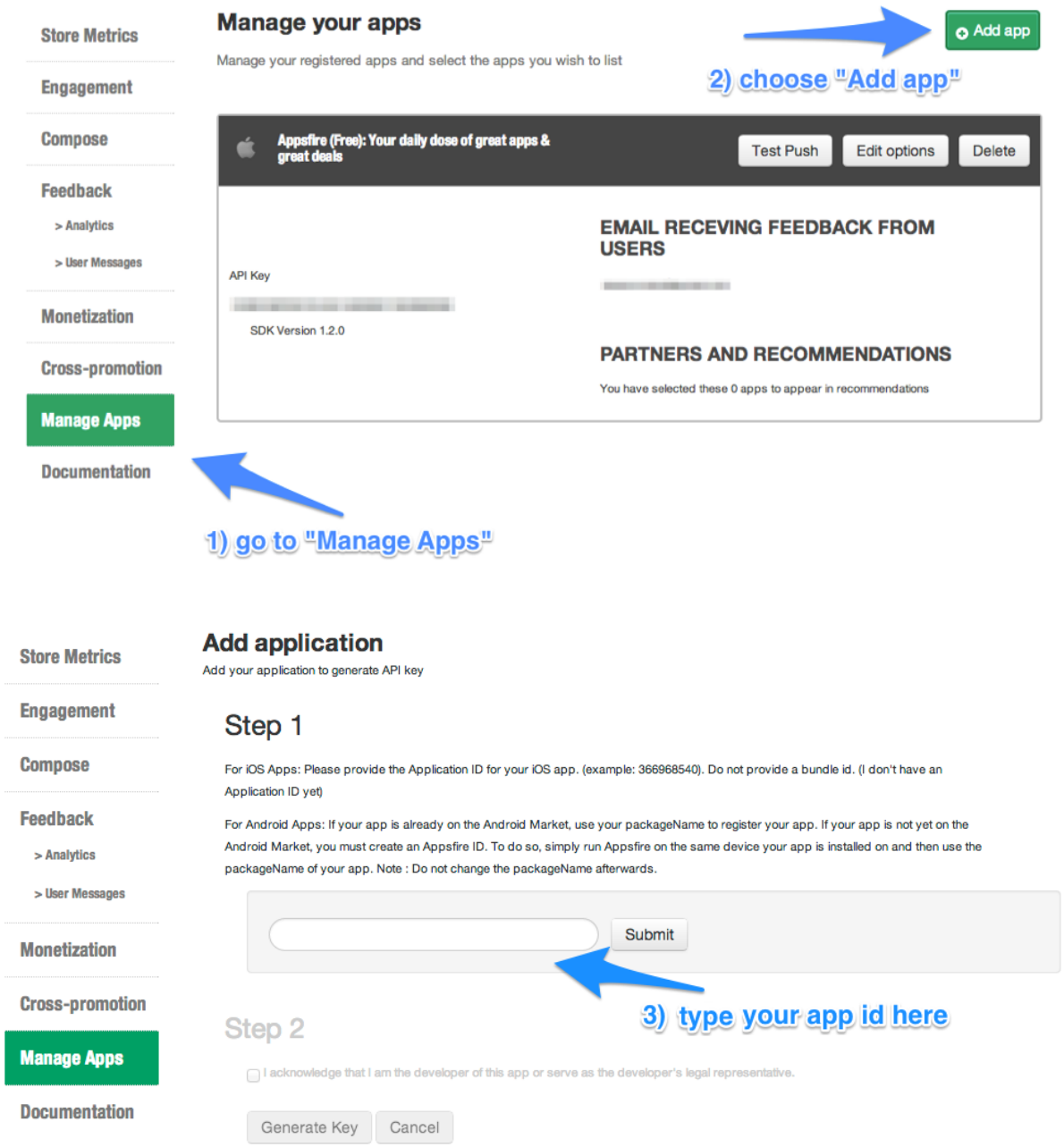
- Pause/Resume functionality (See Advanced) 2) Improved speed
- Lots of stability improvement
- Ground-work for features coming soon…

C. API key

This step is only required for first time integration of the SDK within each of your apps.
If you have already integrated the library in your app, you may skip to the next section.

Please log in to the Dashboard and go to http://dashboard.appsfire.com/app/manage.

Register your application by providing your app id or package name as follows:



By pressing the **Generate Key** button, you will be provided with a unique API key that you need to use in your code.

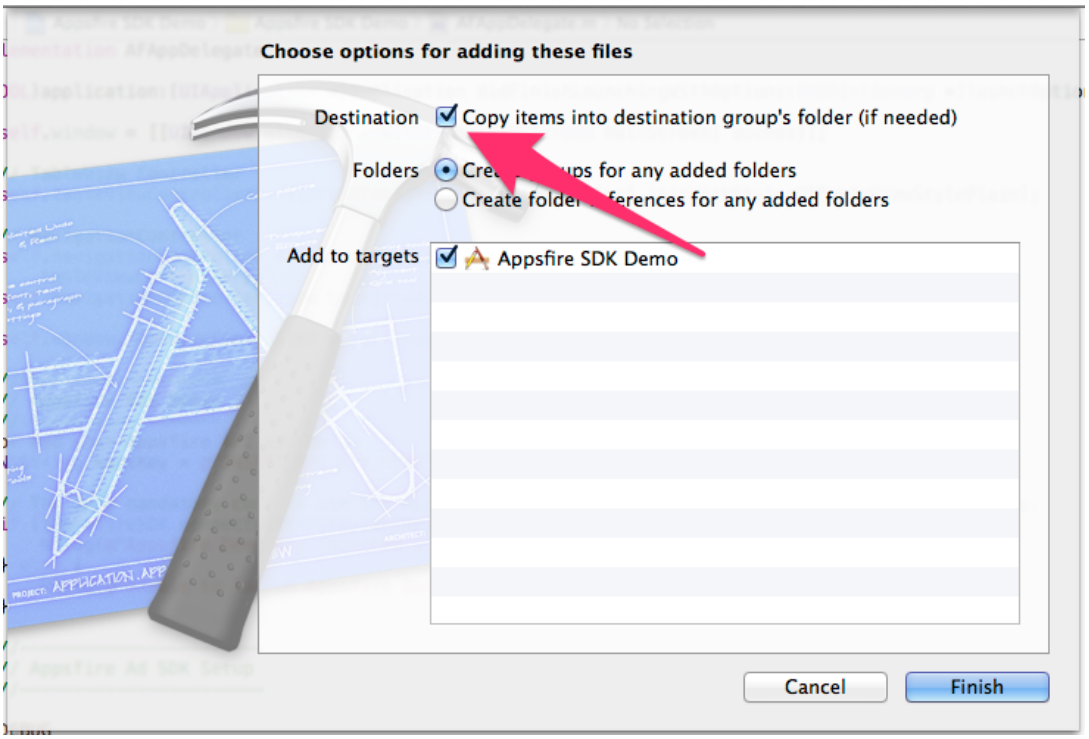
D. Integration in your project

If you are looking for third-party integration (like cross-platform scripting or mediation), please check the Dashboard. A dedicated documentation is included inside each archive file.

Step 1. Drop the assets inside your project

The sample project displays the simplest way to install the SDK inside your app, by drag & dropping the folder “appsfire-sdk” into your own project from the sample project.

Just don’t forget to copy the files into your project by checking this little box:



Step 2. Make sure all frameworks are included

The Appsfire SDK requires the following frameworks to operate. Make sure to have them in your project. **If you omit to do so, your project will not compile and XCode will produce errors**. Some can be checked as “optional” if you are using iOS6+.

- AdSupport (you can check it as “optional”)
- CoreGraphics
- Foundation
- QuartzCore
- StoreKit (you can check it as “optional”)
- SystemConfiguration
- UIKit

Step 3. One-minute implementation of the initialization call

To start with, make sure you import the header file called “AppsfireSDK.h”.

```
#import "AppsfireSDK.h"
```

You have one mandatory call: the connection to our service back-end with your API key.

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

      // connect to appsfire services
      [AppsfireSDK connectWithAPIKey:@"INSERT YOUR API KEY HERE"];

      // [...]
      // window & root controller creation
      return YES;

}
```

Step 4. One-minute implementation of the Engagement features

If you are implementing the following call in a different class than the app delegate, make sure to import the main header file “AppsfireSDK.h”.

```
#import "AppsfireSDK.h"
```

To display the notification wall within your app, you will then just need to call a simple method. We recommend that you call this method as a result of user interaction (i.e. tap of a button).

```
[AppsfireSDK presentPanelForContent:AFSDKPanelContentDefault
withStyle:AFSDKPanelStyleDefault];
```

You can fully customize your integration using various methods illustrated later in this document.

Step 5. One-minute implementation of the Monetization features

This section is required only if you want to monetize your application by displaying ads.
To start with, make sure you import the Advertising header file called “AppsfireAdSDK.h”.

```
#import "AppsfireAdSDK.h"
```

Then, inside the didFinishLaunching call-back method of your application delegate, you can define the various options:

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

      // connect to appsfire services
      [AppsfireSDK connectWithAPIKey:@"INSERT YOUR API KEY HERE"];

      // here you can use the debug mode
      // it'll allow you to see an ad whatever the situation
      // do not ship a build with this debug mode enabled!
      #ifdef DEBUG
      [AppsfireAdSDK setDebugModeEnabled:YES];
      #endif

      // tell advertising sdk to prepare (optional)
      // this method will be automatically called if you do a request
      // you only need to call it one time in your application life
      [AppsfireAdSDK prepare];

      // [...]
      // window & root controller creation
      return YES;

}
```

We recommend that you call the "prepare" method as soon as possible so the library can prepare itself (and preload the corresponding assets). In the worst case, this method will be called upon your actual ad request later on.

The final step involves requesting a modal ad somewhere else in your code. Please make sure to replace the only requested parameter with the proper UIViewController. Most of the time the following example should work. But it all depends on the architecture of your application.
Requesting an ad may not display it right away. Your request is added to a queue and is treated as soon as the ad is available. This is the reason why we recommend that you check its availability before anything else.

```
if ([AppsfireAdSDK isThereAModalAdAvailable]) {

      [AppsfireAdSDK requestModalAdWithController:[UIApplication
sharedApplication ].keyWindow.rootViewController];

}
```

Thanks to a delegate method (see sections below), you can cancel its presentation on the screen if you decide it's not the right time.

We recommend that you read the next section to understand how to be notified when ads are loaded (via the delegate).

E. Integration via the Appsfire SDK Convenience Class

An alternative and very easy way to integrate the Appsfire Engagement and/or Monetization features can be done via the AppsfireConvenienceDelegate class. This is a subclass of

UIResponder and can be used as a superclass of your application delegate. For instance, the SDK Demo Project is using this convenience class.

In this section we assume that you already imported all the necessary files of the Appsfire SDK and added all the Frameworks (as explained above).

To implement it, you first need to import the two following files **`AppsfireConvenienceDelegate.h`** and **`AppsfireConvenienceDelegate.m`** which constitutes the class, then use it as the superclass of your application delegate:

```
#import <UIKit/UIKit.h>
#import "AppsfireConvenienceDelegate.h"

@interface YourAppDelegateClass : AppsfireConvenienceDelegate
// Your properties and methods here.
@end
```

Please note that there is no reference of the typical UIWindow property anymore in your application delegate, it has been moved in the convenience class and you can remove it from your app delegate class.

Basically the convenience class will add all the necessary code to set up the SDK in the following methods:

`-application:didFinishLaunchingWithOptions:launchOptions:`
`-application:didRegisterForRemoteNotificationsWithDeviceToken:deviceToken:`
`-application:didReceiveRemoteNotification:`
`-applicationDidBecomeActive:`

So, in your application delegate, do not forget to call **`super`** when implementing them. For instance, in your implementation file:

```
@implementation AppDelegate

#pragma mark - Monitoring App State Changes

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    [super application:application didFinishLaunchingWithOptions:launchOptions];

    // Your code here

    return YES;
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    [super applicationDidBecomeActive:application];
}


#pragma mark - Handling Remote Notifications

- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken {
    [super application:application
didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo {
    [super application:application didReceiveRemoteNotification:userInfo];
}

@end
```

Finally, if you look at the top of the implementation file (**`AppsfireConvenienceDelegate.m`**), you will see a list of options which will allow you to configure the SDK.

Please take a look at the Demo Project to have an example of the integration.

F. Upgrading from 1.X to 2.0

This part is required only if you already have the Appsfire SDK implemented in your app(s).
A lot of changes and optimizations occurred between 1.1.5 and 2.0.
Overall, you don't have to rethink the way you implemented our library, however please find below the major points you'll have to review for a smooth upgrade.

Class name changed

The main class name was changed from "AFAppBoosterSDK" to "AppsfireSDK". It is therefore safe to do a "*Replace All*" from the old to the new name.

ARC now fully supported

ARC (Automatic Reference Counting) is now fully supported. You need to remove any compilation flag that was previously preventing ARC processing.

Two more frameworks

Please add “AdSupport” and “StoreKit” frameworks into your project.

Deprecated methods

A lot of methods were deprecated. Sometimes because of improvements, other times for better homogeneity. Please find below the list and how you can handle each case in your app.

- Methods which will be removed:
  - **‘abortInitialization’**: we decided to remove this method. If you need at some point to stop activity, take a look at the ‘pause’ method.
  - **‘openUDID’**: OpenUDID was deprecated with iOS6. Furthermore, OpenUDID no longer works as initially designed with iOS7. We are still returning an ID in SDK 2.0, but we’ll completely suppress it in a future iteration.
  - **‘useFullScreenStyle’**: you can specify the kind of display in the presentation method (see below).

- Renamed methods:
  - ‘closeNotifications’ was renamed ‘dismissPanel’
  - ‘setApplicationDelegate:’ was renamed ‘setDelegate:’
  - ‘useCustomValues:’ was renamed ‘setCustomKeysValues:’

- Methods that were merged / updated:
  - ‘presentNotifications’ / ‘presentFeedback’: you should use 'presentPanelForContent:withStyle:' method instead.
  - ‘useGradients:’: we updated our panel to the newer iOS7 look. We do not use gradients anymore. Please take a look at ‘setBackgroundColor:textColor:’ method instead!

Two examples:

If you want to replace the way to display notifications:

```
// old way
// [AFAppBoosterSDK presentNotifications];

// new way
[AppsfireSDK presentPanelForContent:AFSDKPanelContentDefault
withStyle:AFSDKPanelStyleDefault];
```

If you want to customize colors (careful, don’t purely copy/paste the gradients colors into background/text colors, parameters have a different meaning in 2.0):

```
// old way
// [AFAppBoosterSDK useGradients:@[[UIColor colorWithRed:.1137 green:.2274 blue:.7764 alpha:
1.0], [UIColor colorWithRed:.2196 green:.3411 blue:.9333 alpha:1.0]]];

// new way
[AppsfireSDK setBackgroundColor:[UIColor colorWithRed:66.0/255.0 green:67.0/255.0 blue:
69.0/255.0 alpha:1.0] textColor:[UIColor whiteColor]];
```

Notification Names

In an attempt to harmonize the code, we renamed most of our notification names. And now you can use a static variable to be sure it's correctly spelled! You can find the list in "AppsfireSDKConstants.h".

Old representation	New representation
@"AFSDKisInitializing"	kAFSDKIsInitializing
@"AFSDKisInitialized"	kAFSDKIsInitialized
@"AFNotificationNotificationsCounterNeedsUpdate"	kAFSDKNotificationsNumberChanged
@"AFSDKdictionaryUpdated"	kAFSDKDictionaryUpdated
@"AFNotificationsHaveBeenLoaded"	kAFSDKPanelWasPresented
@"AFNotificationsHaveBeenUnLoaded"	kAFSDKPanelWasDismissed

If you encounter any problem you can't solve, don't hesitate to contact us (see "Support Considerations" section).

II. Appsfire Engagement Features

A. Guided tour of each method

1. Basics

Initialization of the SDK

Thi is done by calling the **`+connectWithAPIKey:`** method. This is probably the most important call you need to make in order to set up the Appsfire SDK and the API key can be found on your dashboard.

As this method returns a boolean to whether the SDK connection has been correctly made, you'll need to do something like this:

```
if ([AppsfireSDK connectWithAPIKey:@”API_KEY”]) {
	AFSDKLog(@"Appsfire Demo App launched with %@",[AppsfireSDK getAFSDKVersionInfo]);
}
else {
	AFSDKLog(@"Unable to launch Appsfire Demo App. Probably incompatible iOS");
}
```

Delayed Initialization

If your app has a lot of things to load from the web at startup, you may want to delay the initialization of the Appsfire SDK by a few seconds to give your app full bandwidth. You can do this with the following method:

```
// Initialization with a delay of 2 seconds.
if ([AppsfireSDK connectWithAPIKey:@”API_KEY” afterDelay:2.0]) {
	AFSDKLog(@"Appsfire Demo App launched with %@",[AppsfireSDK getAFSDKVersionInfo]);
}
else {
	AFSDKLog(@"Unable to launch Appsfire Demo App. Probably incompatible iOS");
}
```

In the example above, the delay is 2 seconds. It is your responsibility to prevent the user from trying to launch the SDK window prior to the delay period. If a call is made to present the notifications wall, an error message will appear to the user inviting him/her to try again later.

Get events

To have the best possible integration of the SDK, you will need to set a delegate in order to receive important events. **`+setDelegate:`** sets the delegate of the SDK to the class of your choice. The list of the available events is detailed later in this document.

Checking if the SDK is initialized

For some reason you will need to check if the SDK is initialized, this is possible via the **`+isInitialized:`** method which returns a boolean. For instance, you will need to check if the SDK is initialized before presenting the feedback form.

```
if ([AppsfireSDK isInitialized]) {
      // Do what you want knowing that the SDK is initialized.
}
```

Pausing and Resuming the SDK

Appsfire SDK checks for new notifications approximately every two minutes. As of version 1.1.4, you can pause and resume these refresh cycles using :

```
[AppsfireSDK pause];
```

and you can resume the refresh cycles by calling :

```
[AppsfireSDK resume];
```

In general, you don't need to do this but if your app has some CPU intensive processes (video editing, game rendering, etc) and you prefer dedicating all resources to the process, using these methods gives you control over the Appsfire SDK activity.

2. Push Settings

Register the push token for APNS (Apple Push Notification Service)

In order to send Push Text Alerts from the Appsfire SDK dashboard and/or handle your badge count remotely, you will need to register your push token with Appsfire. This is done via the **`+registerPushToken:`** method. You should call this method in the **`-application:didRegisterForRemoteNotificationsWithDeviceToken:`** method of your application delegate :

```
[AppsfireSDK registerPushToken:deviceToken];
```

Local badge handling

You can decide if you want the Appsfire SDK to handle the badge count of your app locally, only on the device and only while the app is alive:

```
[AppsfireSDK handleBadgeCountLocally:YES];
```

Note that **`-handleBadgeCountLocally:`** overrides any settings established by **`-handleBadgeCountLocallyAndRemotely:`**, and vice versa.

Local and Remote badge handling

You can decide if you want the Appsfire SDK to handle the badge count of your app remotely. The Appsfire web services will update the icon at all times, locally and remotely, even when app is closed:

```
[AppsfireSDK handleBadgeCountLocallyAndRemotely:YES];
```

Note that **`-handleBadgeCountLocallyAndRemotely:`** overrides any settings established by **`-handleBadgeCountLocally:`**.

If you set this option to `YES`, you need to provide us with your Push Certificate. For more information, visit your "Manage Apps" section on http://dashboard.appsfire.com/app/manage.

3. Present & Close

Presenting the panel for Notifications / Feedback

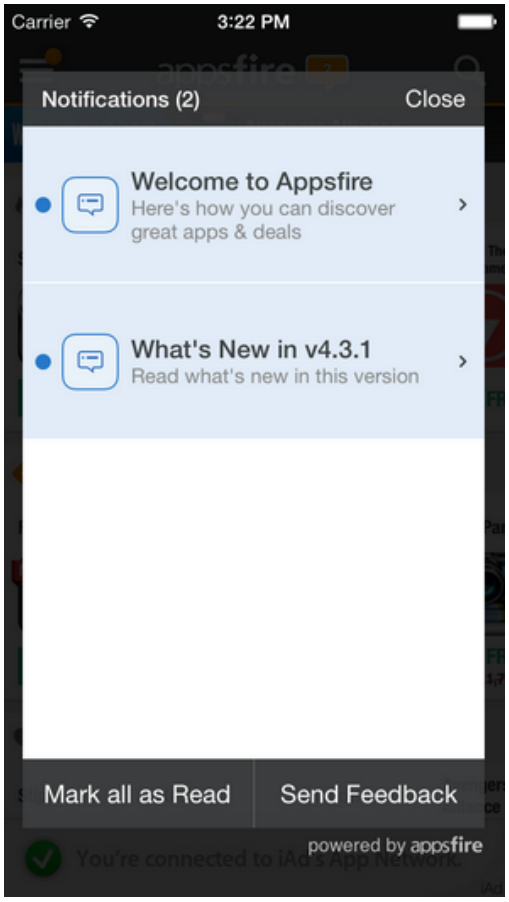
Presenting the SDK Notification Wall or Feedback form is easy:

```
NSError *error= [AppsfireSDK presentPanelForContent:AFSDKPanelContentDefault
withStyle:AFSDKPanelStyleDefault];
```

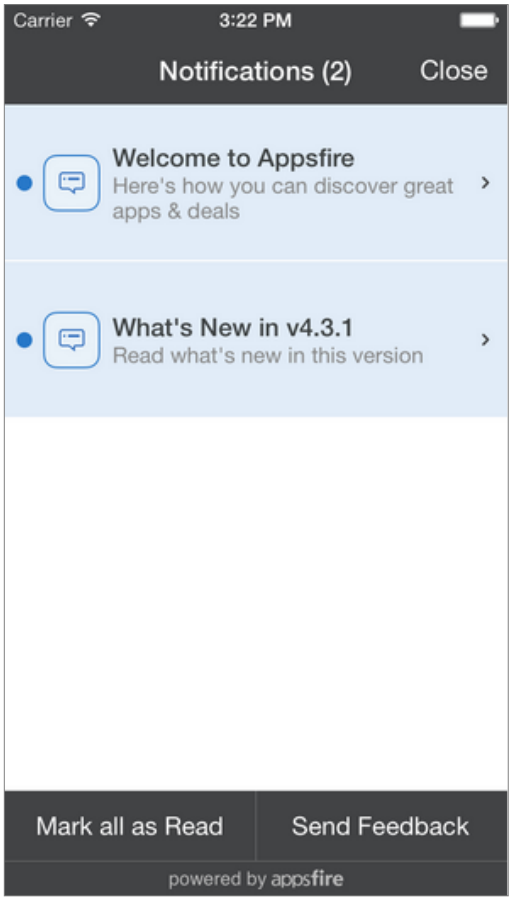
There are several parameters and we will go through the specification details of each of them.

Content types:
- **`AFSDKPanelContentDefault`** : Displays the Notifications Wall
- **`AFSDKPanelContentFeedbackOnly`** : Displays the feedback form only, the notification form won't be visible.

Panel styles:
- **`AFSDKPanelStyleDefault`**: The window will be centered and will not be full screen. So your app remains visible in the back.
- **`AFSDKPanelStyleFullscreen`**: The window will be full screen. Please note that calling the full screen style isn't available on iPad and will force the use of the default style.



AFSDKPanelStyleDefault



AFSDKPanelStyleFullscreen

Closing the Notification Wall / Feedback

The user can close the SDK view by himself via a close button visible at the top right of the window. However if you need to close it by yourself, you can use the following method:

```
[AppsfireSDK dismissPanel];
```

Getting a view controller that contains the notifications view

For more flexibility we are providing a way to get a view controller that contains the notifications view. This will allow you to embed it in a navigation controller or even a tab bar controller:

```
NSError *error;
UIViewController *notificationViewController = [AppsfireSDK
getPanelViewControllerWithError:&error];

if (notificationViewController != nil) {
    [self.navigationController pushViewController:notificationViewController animated:YES];
} else {
    NSLog(@"Error while getting the panel view controller : %@", error.localizedDescription);
}
```

Important: You are responsible for presenting and dismissing the controller. Any leak of this controller could lead to a dysfunction of the SDK. If you encounter a problem in your integration, don't hesitate to contact us (see "Support Considerations" section).

In the example above you can see that we are actually checking if the view controller returned by the **`-getPanelViewControllerWithError:`** method is not nil before presenting the view. This is indeed a necessary check.

Check if Appsfire SDK is visible

Checking if the Notification wall or Feedback form is currently displayed on the screen could be useful, this can prevent many visual problems:

```
[AppsfireSDK isDisplayed];
```

When you are directly handling the UIViewController (via '-getPanelViewControllerWithError:'), it'll return 'YES' whenever the view is on the screen.

Open a specific Notification

You have the ability to open the SDK window initialized on a specific notification, most of the time you will use this when receiving a push alert containing a notification id:

```
NSDictionary *notification = [launchOptions
objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];

if ([notification respondsToSelector:@selector(objectForKey:)]) {
    NSNumber *notificationId = [notification objectForKey:@"ab_notid"];
    if (notificationId != nil && [notificationId respondsToSelector:@selector(intValue)]) {
        AFSDKLog(@"App Launched via Push notification");
        [AppsfireSDK openSDKNotificationID:[notificationId intValue]];
    }
}
```

In the example above we are looking for the **notificationId** and presenting the corresponding notification to the user during the application startup.

4. Customization

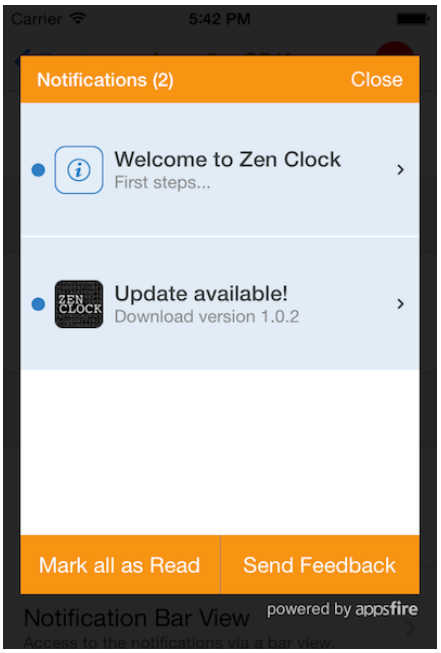
Color customization

You can customize the colors used for the SDK interface. It'll mainly affect the header and the footer of the panel that is presented:

```
[AppsfireSDK setBackgroundColor:[UIColor colorWithRed:247.0/255.0 green:148.0/255.0 blue:
18.0/255.0 alpha:1.0] textColor:[UIColor whiteColor]];
[AppsfireSDK presentPanelForContent:AFSDKPanelContentDefault
withStyle:AFSDKPanelStyleDefault];
```

As you can see the method **`+setBackgroundColor:textColor`** allows you to separately customize the background color and the text color (over the specific background color). Using a **`nil`** value will apply the default colors of the panel (dark gray background color and white text color).

Here is how it looks like once it has been customized with the code above:



Sending substitution data to the SDK

In some cases, you may want to supply the SDK with data. You can do so by sending an NSDictionary of key/value pairs that will then be substituted whenever possible.

Example: You create a notification in the SDK back-office and you want the first line of text to say “Hello Bob, you’re 34 years old” where “Bob” is your user’s first-name and 34 is his age; you’ve collected this information through your own processes. You could do this :

```
NSDictionary *customTags = @{
    @"FIRSTNAME" : @"Bob",
    @"AGE" : @"34"
};
[AppsfireSDK setCustomKeysValues:customTags];
```

By doing this, any instance of the string [FIRSTNAME] or [AGE] will be replaced by the corresponding values (don’t add the brackets in each key).

You can also use this method to generate dynamic URLs.

Setting the User’s Email

If you already know your user’s email address, you can improve the user experience of sending feedbacks by sending it to the Appsfire SDK servers. Doing so, the feedback form will have your user’s email pre-filled.

```
[AppsfireSDK setUserEmail:@"user@email.com" isModifiable:YES];
```

The **isModifiable** parameter will make the email field of the feedback form visible or not. Even if not modifiable, you can set the email so that it appears on the email you receive. This method returns a boolean whether it has correctly been executed or not.

Removing the Feedback button

If you wish to remove the Send Feedback button, you can use this command :

```
[AppsfireSDK showFeedbackButton:NO];
```

Note that you will still be able to show the feedback form via in-app messages or via direct calls to :

```
[AppsfireSDK presentPanelForContent:AFSDKPanelContentFeedbackOnly
withStyle:AFSDKPanelStyleDefault]
```

But the button will not show on the bottom right of the Notification Wall view.

5. Get Information

Appsfire SDK version

Get SDK version and build number. It's only for you to clearly know what version of the SDK you are running. Please send us this string when you contact us, it'll be easier to help you!

```
NSString *version = [AppsfireSDK getAFSDKVersionInfo];
```

Number of notifications

Get the number of pending notifications which require attention. This information is very important in order to notify the user of the available notifications. See the UI Integration section to know how you can take advantage of this information in order to have a good "call-to-action".

```
NSInteger integer = [AppsfireSDK numberOfPendingNotifications];
```

Session Identifier

Get the current SDK session identifier.

```
NSString *sessionId = [AppsfireSDK getSessionID];
```

6. Misc

Resetting the cache (Do not use lightly!)
Resets the SDK cache completely, all user settings will be erased. This includes messages that have been read, icon images, assets, etc.

```
[AppsfireSDK resetCache];
```

If you're having an issue that only this seems to solve, please contact us immediately.

B. Use of Delegate

Prepare your class to support the delegate events

We recommend you to use the Appsfire SDK delegate for an optimal implementation of the library. First you need to add **`AppsfireSDKDelegate`** to the @interface of the chosen class (most of the time it'll be the application delegate):

```
@interface AppDelegate : UIResponder <UIApplicationDelegate, AppsfireSDKDelegate>
// properties & methods
@end
```

Do not forget to set this same class as the delegate via:

```
[AppsfireAdSDK setDelegate:self];
```

Dismissing SDK view controllers

In the case of a view controller fetched with **-getPanelViewControllerWithError:** you need to manually dismiss it and this is possible via the following delegate method **-panelViewControllerNeedsToBeDismissed:**

For instance, for a view controller pushed in a navigation controller, you might want to do this:

```
- (void)panelViewControllerNeedsToBeDismissed:(UIViewController *)controller {
    [navigationController popViewControllerAnimated:YES];
}
```

or for a view controller modally presented:

```
- (void)panelViewControllerNeedsToBeDismissed:(UIViewController *)controller {
    [controller dismissModalViewControllerAnimated:YES];
}
```

Notification Opening Callback

The SDK provides a callback to the **-openSDKNotificationID:** which allows you to know if the the specific notification has been correctly opened. If an error occurred while fetching the notification, you can either display a message to the user or simply retry.

```
- (void)openNotificationDidFinish:(BOOL)succeeded withError:(NSError *)error {
    if (!succeeded) {
        NSLog(@"An error occurred while opening the notification: %@",
error.localizedDescription);
    }
}
```

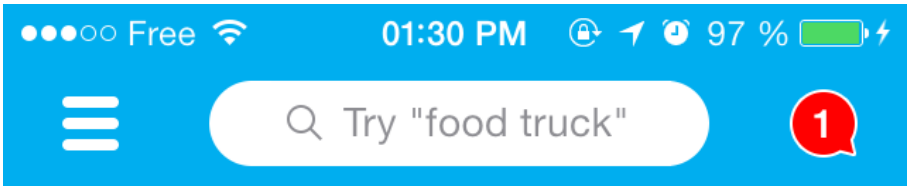
C. UI Integration

1. Adding your “Call-to-Action”

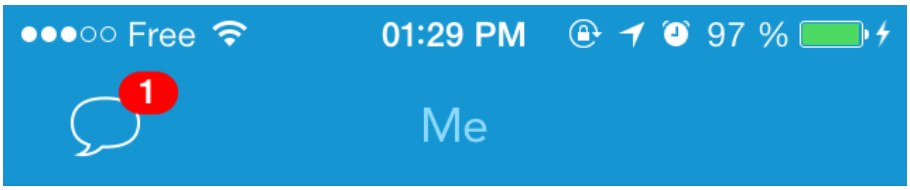
Before we elaborate on best practices, please note that as part of the Appsfire SDK service, we are happy to assist you in determining the best placement. Our product designer (jonathan@appsfire.com) is available by email or Skype to consult with you. We are even able to supply you with icons and/or buttons that will suit your needs. Please feel free to reach out.

Through experience in our apps, measurement of the alpha-stage Appsfire SDK adopters, and observation of the latest trends in mobile UI/UX, it’s clear that the best places to place an engagement-based call-to-action in your app are in the navigation bar or the tab bar. The objective is to give users a prominent and reliable place to go to get the latest notifications from your app so that as soon as they open the app, they know just where to go. And for uninitiated users, the call- to-action will be impossible to miss.

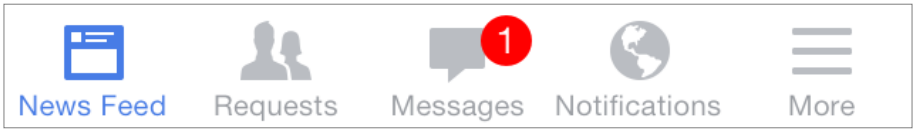
Here is list of examples to illustrate this:



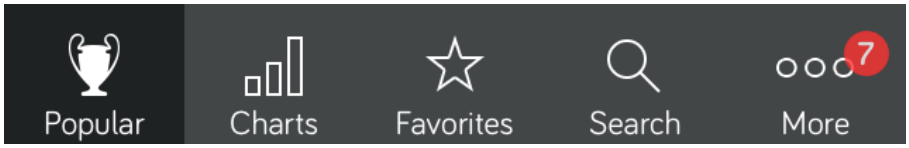
Foursquare - Navigation Bar



RunKeeper - Navigation Bar



Facebook - Tab Bar



Appstatics - Tab Bar

The Appsfire SDK package provides an out-of-the-box solution to display an SDK notification in your app. Refer to the section for the integration of this component in your app.

2. Notification Badge View

The notification badge is a simple circular view with a count in it that reflects the number of notifications which require your attention. Here is a preview of the default look:

The integration is very simple, first you need to import the **`AFBadgeView.h`** and **`AFBadgeView.m`** in your project. Then you simply instantiate and add it to a view with the following lines:

```
AFBadgeView *badgeView = [[AFBadgeView alloc] initWithFrame:CGRectMake(0, 0, 30.0, 30.0)];
[view addSubview:badgeView];
[badgeView updateBadgeCount];
```

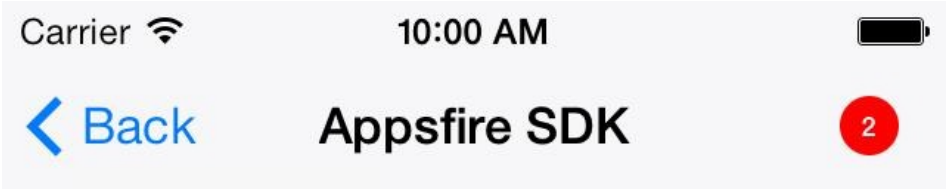
You have the ability to customize the look of this badge, here is an example with a code sample:


```
// Change the color of the badge.
badgeView.backgroundView.backgroundColor = [UIColor colorWithRed:0.0 / 255.0 green:114.0 /
255.0 blue:226.0 / 255.0 alpha:1.0];

// Change the “roundness” of the badge.
badgeView.backgroundView.layer.cornerRadius = 4.0;

// Change the font of the count.
badgeView.textLabel.font = [UIFont fontWithName:@"Avenir-Medium" size:13.0];
```

For instance, if you want to add this view to the navigation bar, here is a simple code snippet to help you do that:

```
- (void)viewDidLoad {
    [super viewDidLoad];

    // Title
    self.title = @"Appsfire SDK";

    // Creating the badge
    AFBadgeView *badgeView = [[AFBadgeView alloc] initWithFrame:CGRectMake(0.0, 0.0, 20.0,
20.0)];

    // Adding a target to the touched event.
    [badgeView addTarget:self action:@selector(didTouchBadgeView:)
forControlEvents:UIControlEventTouchUpInside];

    // Manually update the badge.
    [badgeView updateBadgeCount];

    // Add it to the right of the navigation bar.
    UIBarButtonItem *badgeButtonItem = [[UIBarButtonItem alloc]
initWithCustomView:badgeView];
    self.navigationItem.rightBarButtonItem = badgeButtonItem;
}

- (void)didTouchBadgeView:(id)sender {
    [AppsfireSDK presentPanelForContent:AFSDKPanelContentDefault
withStyle:AFSDKPanelStyleDefault];
}
```

Here is what it looks like:



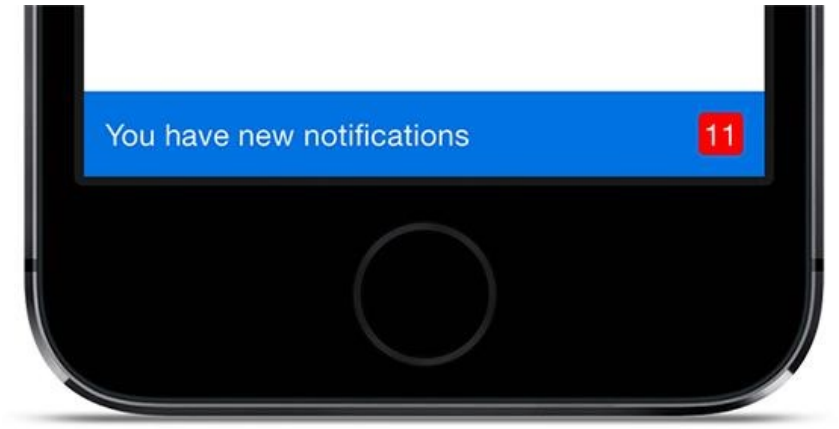
We recommend that you take a look at the header file for a complete overview of properties and methods.

3. Notification Bar

The Appsfire SDK package bundles an out-of-the-box solution in case the navigation bar or the tab bar are not the right placement for your call-to-action.

The notification bar is a subtle bar that will appear on the bottom of the user's device or just above toolbars and tab bars and will provide your users with quick and easy access to the Notification Wall. The text of the bar is automatically updated and translated based on the user's

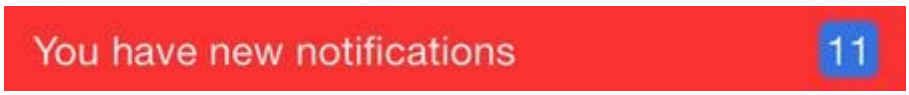
preferences and the number of unread notifications appears in the badge on the right. Here is a preview:



To implement it in your project, you first need to import the **`AFNotificationBarView.h`** and **`AFNotificationBarView.m`** files and instantiate it:

```
AFNotificationBarView *notificationBar = [[AFNotificationBarView alloc]
initWithContainerView:self.view displayMode:AFNotificationBarViewStyleBottom];
```

Customization is also possible, here is an example:



And the associated code implementation:

```
// Changes the "roundness" of the count label.
notificationBar.countLabel.layer.cornerRadius = 4.0;

// Changes the color of the notification bar.
notificationBar.backgroundColor = [UIColor colorWithRed:255.0 / 255.0 green:50.0 / 255.0
blue:50.0 / 255.0 alpha:1.0];
notificationBar.countLabel.backgroundColor = [UIColor colorWithRed:0.0 / 255.0 green:114.0 /
255.0 blue:226.0 / 255.0 alpha:1.0];

// Changes the height of the notification bar.
notificationBar.barHeight = 30.0;
```

As for the Notification Badge, we recommend you to take a look at the header file for a complete overview of the properties and methods.

For both libraries mentioned above, there are examples of usage in the SDK Demo Project, please refer to it.

D. Notifications

The Appsfire SDK fires several notifications you can observe at any place of your application.

SDK is initializing
Observer name: `kAFSDKIsInitializing`
Fired once when the library starts the initialization.

SDK is initialized
Observer name: `kAFSDKIsInitialized`
Fired once when the library finished the initialization.

Notifications count was updated
Observer name: `kAFSDKNotificationsNumberChanged`
Fired each time the notification count is changed. This allows you to know when to update any view displaying this number (like a badge).

Localized strings were updated
Observer name: `kAFSDKDictionaryUpdated`
Fired when the strings used in the sdk were updated from our web-service. The library contains some defaults key/values for several languages. We load and update more strings depending on the device of your users!

Panel was presented
Observer name: `kAFSDKPanelWasPresented`
Fired each time the (notifications / feedback) panel is presented on the screen.

Panel was dismissed
Observer name: `kAFSDKPanelWasDismissed`
Fired each time the (notifications / feedback) panel is dismissed from the screen.

Example: you can observe an event like this:

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(methodName)
name:kAFSDKIsInitializing object:nil];
```

III. Appsfire Monetization Features

A. Full tour of methods

1. General Options

Using the in-app overlay

In iOS 6 it becomes possible to display an item page in the App Store via a modal controller instead of redirecting the user to the App Store app. This is enabled by default in the Advertising SDK, as it serves you and your users by not kicking them out of your app.

If the user is on iOS 5, or if a problem occurs with the in-app overlay (this particular Apple API isn't very stable), then the user will be redirected to the App Store app.

You can modify the default value (`YES`) like this:

```
[AppsfireAdSDK setUseInAppDownloadWhenPossible:YES];
```

Test your implementation via the "debug" mode

We added a debug mode because there may not be an ad available when you implement the Monetization features in your application or simply for testing purpose.

In debug mode:
- the SDK will display an app no matter what. Chances are it'll be the Appsfire/Appstatics app.
- there won't be any capping here. This means that you'll be able to display the ad as much as you want (vs. once a day or capped while in production mode).
- you probably won't have any stats in the dashboard while the debug mode is enabled.

Be careful, though. **Do not keep Debug Mode on** in your app when submitting to iTunes Connect. We recommend that you implement this debug mode as follows (keep the #if #endif as a safeguard):

```
#if DEBUG
	[AppsfireAdSDK setDebugModeEnabled:YES];
#endif
```

2. Modal Ad Methods

Request a modal ad

As mentioned previously, you can easily request a modal ad. You only need to pass a UIViewController parameter that will be used to present the modal, and eventually the in-app overlay.

Important: this method isn't necessarily synchronous. If ads aren't loaded when you call the method, your request will be added in a queue and processed a bit later.

```
[AppsfireAdSDK requestModalAdWithController:[UIApplication
sharedApplication].keyWindow.rootViewController];
```

In our example we use the "rootViewController" of the "keyWindow" because it'll be correct for most of the applications. Be sure to indicate the proper UIViewController depending on your application.

Check if there is a modal ad to display

If you want to check the availability of the modal ad in a breakout session, there is a straightforward method for that. Not only it'll verify if the library is correctly initialized and ads are loaded from the web service, but it will also check if a modal ad can be displayed right now.

```
if ([AppsfireAdSDK isThereAModalAdAvailable]) {
      // request a modal ad here
}
```

Cancel a pending request

Your request won't necessarily display a modal ad as soon as you request it (e.g., library isn't initialized, ads aren't loaded yet). In this case, your request will be added in a queue and will be processed at the right moment. If, in the meantime, you decide to abort any pending request, you can use the following method:

```
[AppsfireAdSDK cancelPendingAdModalRequest];
```

If a modal ad is canceled, the method will return `YES` and you'll get a delegate event via the `modalAdRequestDidFailWithError:` method.

Check if a modal ad is currently displayed

We implemented this as an easy way for you to check if the SDK is currently displaying something on the screen.

```
BOOL isModalDisplayed = [AppsfireAdSDK isModalAdDisplayed];
```

Note that you can know the start and the end of a modal being displayed via the Advertising delegate.

B. Use of delegate

Prepare your class to support the delegate events

We recommend that you use the Advertising delegate for a better implementation of the library. First you need to add “AppsfireAdSDKDelegate” to the @interface of the chosen class (most of the time it’ll be the app delegate):

```
@interface AppDelegate : UIResponder <UIApplicationDelegate, AppsfireAdSDKDelegate>
// properties & methods
@end
```

Don’t forget to set this class as the delegate in the “didFinishLaunchingWithOptions” method:

```
[AppsfireAdSDK setDelegate:self];
```

Once this is done, you’ll be able to receive various events:

All are optional, you can implement the ones you want. All these calls are done on the main thread.

1. When the library is initialized

This method isn’t critical for a basic implementation. But you will know that the library initialized correctly (but didn’t receive any ad yet).

```
- (void)adUnitDidInitialize {

}
```

2. When the first modal ad is ready to be displayed after the library initialization

This method is very useful for a basic implementation. You can use it to fire an internal event in your application, so you know there is an ad to display.

```
- (void)modalAdIsReadyForRequest {

	// if you request the ad immediately, you can be sure that a modal ad is
available
	// you need to specify the UIViewController that will be used to display
the modal add & eventually the in-app overlay
	[AppsfireAdSDK requestModalAdWithController:[UIApplication
sharedApplication].keyWindow.rootViewController];

	// if it's not the proper time, then you can request it a bit later

}
```

3. If a modal overlay fails to present

This method is not necessary in your application's life, but can be useful to debug an eventual problem.

It's called when the modal ad you requested could not be displayed. There are various reasons this could occur, and you can use the code in the "NSError" to understand why. The most common one is "AFSDKErrorCodeAdvertisingNoAd", meaning there was no ad to display (i.e., it's not a technical issue).

```
- (void)modalAdRequestDidFailWithError:(NSError *)error {

	NSLog(@"Ad Unit Request Failed = %@", error.localizedDescription);

	// optional, you can implement a reaction
	switch (error.code) {
		case AFSDKErrorCodeAdvertisingBadCall:
			break;
		case AFSDKErrorCodeAdvertisingNoAd:
			break;
		case AFSDKErrorCodeAdvertisingAlreadyDisplayed:
			break;
		default:
			break;
	}

}
```

4. When a modal ad is going to be presented

Depending on your implementation of the Monetization features, you may want to cancel the presentation of the modal ad at the very last moment.

If you don’t implement the following method, the modal ad will directly go on screen when ready. If you decide to cancel it because you are doing something important on screen, or if you aren’t in a break-out session, then return `NO`

```
- (BOOL)shouldDisplayModalAd {
      return YES;
}
```

5. When the modal ad will/did appear and will/did disappear

It’s always better to be alerted of tierce library actions. That’s why we tell you when we are going to present the modal ad, and when it’s going to be dismissed.

```
- (void)modalAdWillAppear {
      //
}

- (void)modalAdDidAppear {
      //
}

- (void)modalAdWillDisappear {
      //
}

- (void)modalAdDidDisappear {
      //
}
```

C. Implementation best practices

You have different ways to display ads depending your kind of application. Please find below two cases that covers most of class integrations.

Case 1: Display as soon as possible

A classic implementation is to present the ad when your app becomes active. Just be sure to check that an ad is available, and none is currently displayed.

```
- (void)applicationDidBecomeActive:(UIApplication *)application {

      // check if there is an ad available, and that none is currently displayed
      if ([AppsfireAdSDK isThereAModalAdAvailable] && ![AppsfireAdSDK isModalAdDisplayed]) {

            // delay a bit the request because ‘applicationDidBecomeActive:’ could prevent a
part of the animation
            double delayInSeconds = 0.5;
            dispatch_time_t popTime = dispatch_time(DISPATCH_TIME_NOW, (int64_t)
(delayInSeconds * NSEC_PER_SEC));
            dispatch_after(popTime, dispatch_get_main_queue(), ^(void){
                  [AppsfireAdSDK requestModalAdWithController:[UIApplication
sharedApplication].keyWindow.rootViewController];
            });

      }

}
```

And we recommend you to add the similar code whenever the delegate method “modalAdIsReadyForRequest” is called. You can be sure an ad is available.

```
- (void)modalAdIsReadyForRequest {

      // check if there is an ad available, and that none is currently displayed
      if ([AppsfireAdSDK isThereAModalAdAvailable] && ![AppsfireAdSDK isModalAdDisplayed]) {

            [AppsfireAdSDK requestModalAdWithController:[UIApplication
sharedApplication].keyWindow.rootViewController];

      }

}
```

You can of course decide to display the ad in a more subtle way, like presented in case 2.

Case 2: Display in a breakout session

A breakout session is a natural “break” in the user experience of your application app. It’s a good way to display ads if you don’t want to bother your users too much. For example if you have a game you could display an ad after a game over.

Like the code you saw above, you don’t have much to write! We suggest that you create a method that you’ll be able to call from different places of your application. Then, it’s up to you to call the method when you decide it should be a good time.

```
- (BOOL)displayModalAdIfPossible {

	// if there is no ad or if one is already displayed, stop there!
	if (![AppsfireAdSDK isThereAModalAdAvailable] || [AppsfireAdSDK isModalAdDisplayed])
		return NO;

	// else, request the ad that should quickly appears
	[AppsfireAdSDK requestModalAdWithController:[UIApplication
sharedApplication].keyWindow.rootViewController];

	return YES;

}
```

This example returns a BOOL that you can use to know if an ad is going to be displayed.

D. Find your problem step-by-step

Here is a checklist to help you find an eventual problem you could encounter while implementing the Monetization features into your application. Before contacting us, please check each point so we can help you faster in your debugging process.

1. Is the library correctly initialized?

Before anything, the library needs to initialize itself. Most of the time it'll be done in the first two seconds. But here's how to check if the process was successful:

```
if ([AppsfireAdSDK isInitialized]) {
	NSLog(@"Advertising SDK is initialized");
}
```

It's best to implement the delegate method which will let you know when the library initialized:

```
- (void)adUnitDidInitialize {
   NSLog(@"Ad Unit is initialized");
}
```

Why is it that the library doesn't initialize?

- The process isn't finished.
- You haven't set an API key (in basic SDK requirements).
- The device isn't connected to the internet.

2. Are the ads loaded from the web service?

The second thing you should verify is the connection between the library and the web service. When the library is initialized, we try to fetch the latest ads from the web service. Once received, we'll process any eventual pending request.

Like for the library initialization, you can check if this process was successful. This doesn't mean there'll be an ad available, but only that the synchronization with the server was correctly done.

```
if ([AppsfireAdSDK areAdsLoaded]) {
	NSLog(@"Ads correctly loaded from web service");
}
```

Why is it that the ads will not load?

- Library isn't initialized.
- The process isn't finished.
- The device isn't connected to the internet.

3. Is there a modal ad to display?

Now you are sure ads are correctly loaded, you can check if there is any modal ad available. As for the library initialization, you have two ways to know it.

The first one is pretty simple and can be done whenever you want.

```
if ([AppsfireAdSDK isThereAModalAdAvailable]) {
	NSLog(@"A Modal Ad is Available");
}
```

It's best to implement the delegate method (explained in the previous section) which will let you know when ads are loaded and there is a modal ad available:

```
- (void)modalAdIsReadyForRequest {
	NSLog(@"A Modal Ad is Available");
}
```

Why is it that there is no modal ad available?

- Ads aren't loaded from the web service.
- You aren't in "debug mode" and there is no ad to display today.

If you want to test your application, be sure to be in "debug mode" that will display an app whatever the conditions. Just make sure to disable "debug mode" when submitting your app to Apple!

4. What if requesting a modal doesn't work?

First be sure to implement the delegate method that alerts you when a modal failed to display. It may give you a clue! Check the method "modalAdRequestDidFailWithError:" in section B.

If you verified that a modal ad is available, but nothing happens when you request it, then it's probably a problem with the UIViewController parameter that you are sending. By default our example uses the following controller:

"[UIApplication sharedApplication].keyWindow.rootViewController"

which is the main controller given to the window in the app initialization. But depending on the age of your application (i.e., the iOS version in which it was created), or even its architecture, this may not be the best choice.

In summary, verify that the controller is appropriate. How? Try to present an empty controller to see if something happens on the screen. If not, then it's definitely not the right one.

5. Still a problem?

Don't hesitate to contact us (see "Support Considerations" section), we'll do our best to help you.

IV. F.A.Q. and Support

A. Support Considerations

Please contact jonathan@appsfire.com for general inquiries.

High-level inquiries should be directed to both yann@appsfire.com and ouriel@appsfire.com.

B. Frequently Asked Questions

Base SDK

Feedback Emails are showing “v(null)” or just “v” instead of my app’s version number, Feedback Emails are not showing the right version label

Make sure that you have supplied a value for the key “Bundle versions string, short” in your main info.plist file.

Advertising SDK

How many times a day can the “modal ad” be displayed?

In a normal state, the “modal ad” will be displayed once a day. For example, if a user sees an ad at 11pm, he’ll be able to see the next one at midnight.
However, to help you implement the library in your application, the “debug” mode won’t be affected by this capping system.

What happens if a modal ad is being displayed and app goes to background?

We dismiss the ad and the eventual in-app overlay that were displayed. That way the user won’t be stuck in an ambiguous state.

Can I stop the ads server-side?

In case you would like to stop presenting ads without sending another build to Apple, you have a checkbox in your Dashboard, section Manage Apps, which allows you to do so!