



# AdFalcon iOS Native Ad SDK 4.2.0 Developer's Guide

**AdFalcon Mobile Ad Network**

Product of

**Noqoush Mobile Media Group**



## Table of Contents

1	Introduction .....	4
	Native Ads Overview .....	4
	Prerequisites .....	4
2	Install AdFalcon SDK .....	5
	2.1 Use CocoaPods (Recommended) .....	5
	Initialize the CocoaPods.....	5
	Modify the Podfile .....	5
	Install the pod.....	5
	Update the pod.....	5
	2.2 Add AdFalcon SDK framework manually.....	5
3	Configure Xcode .....	6
	Add “-ObjC” to Other Linker Flags.....	6
	Turn Off App Transport Security (ATS).....	6
4	Native Ad Integration Steps .....	7
	Step 3: Creating Native Ad Template View .....	7
	Step 3.1 Create Native Ad Template View Layout.....	7
	Step 3.2 Adopting ADFNativeAdTemplate Protocol.....	12
	Step 4: Requesting a Native Ad .....	14
	Step5: (Optional) Tracking Ad Lifecycle Events – ADFNativeAdDelegate Protocol.....	15
5	Adding Native Ad to UITableView .....	16
	Step 1: Add ADFHelpers classes to your project. ....	16
	Step 2: Open Your ViewController’s header file. ....	16
	Step 3: Create Your Template View .....	16
	Step 4: Open Your ViewController’s Implementation file.....	16
	Step 5: Replace UITableView’s methods.....	17
5	Adding Native Ad to UICollectionView.....	19
	Step 1: Add ADFHelpers classes to your project. ....	19
	Step 2: Open Your ViewController’s header file. ....	19
	Step 3: Create Your Template View .....	19
	Step 4: Open Your ViewController’s Implementation file.....	19
	Step 5: Replace UICollectionView’s methods .....	19
6	Appendix .....	22
	ADFUserInfo Class .....	22
	ADFTargetingParams Class.....	23
	ADFNativeAd Class .....	23
	ADFNativeAdTemplate Protocol .....	24
	ADFNativeAdDelegate Protocol .....	25



ADFNativeAdError Enum.....	26
ADFNativeAdBinder Class.....	26
7 More Information:.....	29



# 1 Introduction

AdFalcon iOS SDK Native Ad integration guide contains all the information needed by iOS developers to integrate with AdFalcon Network to implement native ads in their apps. The guide will also provide examples and code snippets on how to perform the integration with ease.

## Native Ads Overview

Native ads are a form of digital advertising where the ad experience follows the natural form and function of the user experience in which it is placed.

Unlike standard ads, where AdFalcon SDK displays the ad, when loading native ads AdFalcon SDK returns a native ad object with the ad's assets and the app itself is responsible for displaying the ad. With Native Ads, you control the look and feel, design, layout and position of the ad to seamlessly be an integral part of your app.

## Prerequisites

- Xcode: 8.0 or higher
- Development target: iOS 8.0 or higher
- CocoaPods to manage library dependencies with your Xcode projects (Recommended). for more information about cocoaPods visit [cocoapods site](#)

**Note:** *If you need to support iOS 7, you can download the old SDK via [AdFalcon iOS SDK v3.5.0](#).*



## 2 Install AdFalcon SDK

There are two ways to install our SDK:

### 2.1 Use CocoaPods (Recommended)

To use the CocoaPods you should implement the steps below, For more info visit: <https://cocoapods.org/pods/AdFalconSDK>

#### Initialize the CocoaPods

The below steps illustrate the needed steps to initialize and prepare the CocoaPods.

- Open the terminal app from your applications
- If you haven't installed the CocoaPods before, run this command from the terminal: **sudo gem install cocoapods**
- Go to your project's directory by running this command: **cd [path of your project]**
- Create CocoaPods to your project by running this command: **pod init**
- Open your project's directory, you will find the CocoaPods project was installed and ready.

#### Modify the Podfile

After creating the CocoaPods project, you will find Podfile within the project's directory. Open it via TextEdit or Xcode, then insert the below pod (**pod 'AdFalconSDK'**) to your target

```
platform :ios, '8.0'

target 'YourProjectTarget' do
  pod 'AdFalconSDK'
end
```

#### Install the pod

Run the below command to install AdFalcon's pod to your project.

**pod install**

#### Update the pod

To be sure to get the latest iOS SDK into your project, run the below command from the terminal in the same path of Podfile' directory.

**pod update**

### 2.2 Add AdFalcon SDK framework manually

Follow the procedure below to manually add AdFalcon SDK framework to your project:

- Download [AdFalcon iOS SDK](#).
- Unzip the downloaded zip file, you will find AdFalconSDK.framework in the Framework folder.
- Add the framework to your "Link Binary With Libraries" in "Build Phases" of your project's target.



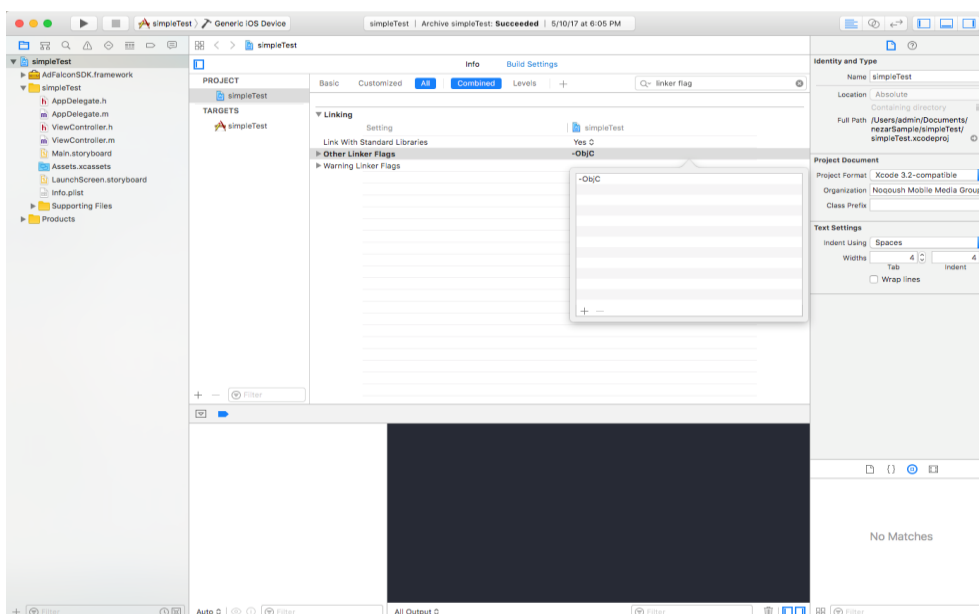
## 3 Configure Xcode

### Add "-ObjC" to Other Linker Flags

You must add -ObjC linker flag to your project; if you forget adding the linker flag, the "Selector not recognized" exception will be raised during the runtime.

follow these steps to add -ObjC:

1. In Xcode, choose View > Navigators > Show Project Navigator, or press  $\mathbb{A}$ 1.
2. Select your project under the PROJECT heading in the Project Navigator, then select the Build Settings tab.
3. Scroll down to the Other Linker Flags build setting under the Linking collection, or type "Other Linker Flags" into the search bar.
4. Set the value of the Other Linker Flags build setting to -ObjC.



### Turn Off App Transport Security (ATS)

If you build your app with iOS SDK 9.0 or later, you will need to turn off App Transport Security in your app's plist in order to complete AdFalcon SDK integration successfully as below:

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSAllowsArbitraryLoads</key>
  <true/>
  <key>NSAllowsArbitraryLoadsForMedia</key>
  <true/>
  <key>NSAllowsArbitraryLoadsInWebContent</key>
  <true/>
</dict>
```



## 4 Native Ad Integration Steps

AdFalcon SDK aims to make native ads implementation in your app a straightforward process where you control the look and feel, design and placement of the ad while AdFalcon SDK handles the rest of the operations needed to display the ad and process impressions and clicks successfully.

When loading native ads, AdFalcon SDK returns a native ad object with the ad's assets, and the app itself is responsible for displaying the native ad by providing a template UIView that lays out the native ad assets in the desired design and look and feel. This process is achieved by mapping the different UIViews (icon, title, description...etc.) in the template to the native ad assets.

The sections below provide a step-by-step guide for native ad implementation.

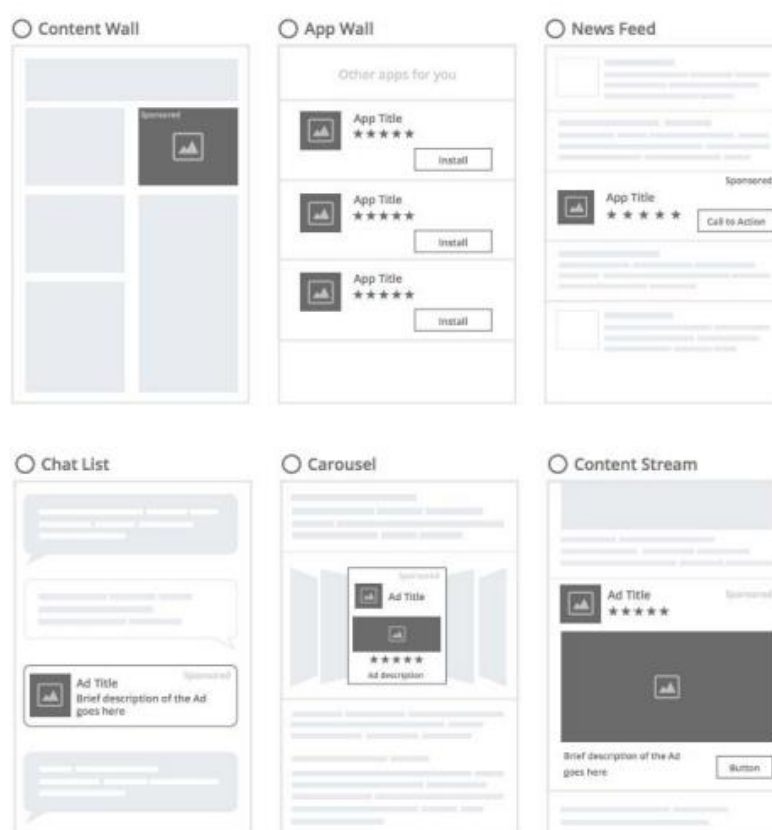
### Step 3: Creating Native Ad Template View

In this step, you create your native ad template view, this template must extend **UIView** and adopt **ADFNativeAdTemplate** Protocol.

#### Step 3.1 Create Native Ad Template View Layout

The native ad template view consists of multiple UIViews, these UIViews are mapped to the native ad assets. You should include in the template only the native ad assets that are needed as per the desired design.

Some examples for native ad templates are shown below.





You can state what native ad assets are requested by including them in the native ad template view. AdFalcon will only return native ads that include the requested assets as per the below rules:

- **Required Assets**

Your native ad must display **AdChoices** view and at least one of the assets below otherwise it will be considered as invalid template:

- **Icon**
- **Title**
- **MainAsset**

AdFalcon will only return ads which contain **all the required assets** included in the native ad template

- **Optional Assets**

All remaining native ad asset types are optional.

The template may include one or more of the optional assets or none at all as per the desired design.

The returned native ad may or may not include any of the requested optional assets; i.e. the optional native ad assets will only be returned when available.

If a native ad response does not contain any of the optional assets, the SDK will hide its associated view.

The below table illustrates all the supported native ad assets which can be included in the template view:

Asset	Required	View Type	Description
AdChoices	Yes	UIView	<p>An AdChoices icon must be displayed in your ad to enable the user to learn about interest-based advertising.</p> <p>The SDK will decide when it is needed to display the icon and will also handle the tap event on the view by opening an opt-out page in the browser.</p> <p><b>Parameters:</b> The size of the view: <b>20x20</b>.</p> <p>You must place the view in any one of the four corners of the ad template; upper right hand corner is recommended.</p>
Title	At least one of Title, Icon or MainAsset must be included	UILabel	<p>String representation of the native ad, which could be the name of the product or the service being advertised.</p> <p><b>Parameters:</b> Maximum title length. The Title UILabel is used to infer the maximum title length, unless it is explicitly set while binding</p>





the template view to the native ad asset.

It is recommended that the UILabel is laid out to display at least **25 characters**.

Only native ads with title assets whose length is less than the requested maximum title length will be returned.

Icon	At least one of Title, Icon or MainAsset must be included	UIImageView	<p>A squared image that shows the icon of the product, service, brand, ... etc.</p> <p><b>Parameters:</b> Minimum width and height. The Icon UIImageView is used to infer the minimum width and height, unless it is explicitly set while binding the template view to the native ad asset</p> <p>Recommended minimum width and height of the Icon UIImageView is <b>50x50</b>.</p> <p>Only native ads with icon assets whose width and height is less than the requested minimum icon width and height will be returned.</p>
MainAsset	At least one of Title, Icon or MainAsset must be included	UIView	<p>The Native Ad main asset can be one of the below types:</p> <ul style="list-style-type: none"> <li>• Image</li> <li>• XHTML (HTML+JavaScript)</li> <li>• Video</li> </ul> <p>The MainAsset is a UIView which is used as a container view for the actual native ad main asset. The SDK will automatically create the adequate UIView type that match the native ad main asset.</p> <p><b>Parameters:</b> Minimum width and height. The Main Asset UIView is used to infer the minimum width and height, unless it is explicitly set while binding the template view to the native ad asset</p> <p>The main asset UIView width and height aspect ratio should always be W/H=1.91. It is recommended to use 320x167 points.</p> <p>Only native ads with main assets whose width and height is greater than the requested minimum main asset width and height will be returned.</p>
Sponsored	No	UILabel	"Sponsored By" message where the response



			<p>should contain the brand name of the sponsor.</p> <p><b>Parameters:</b> Maximum sponsored length. The Sponsored UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>10 characters</b>.</p>
Description	No	UILabel	<p>Descriptive text associated with the product or the service being advertised.</p> <p><b>Parameters:</b> Maximum description length. The Description UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>100 characters</b>.</p>
Star Rating	No	UIImageView	<p>Consists of five stars, which represents the rating of an app.</p> <p><b>Parameters:</b> Minimum width and height. The Star Rating UIImageView is used to infer the minimum width and height, unless it is explicitly set while binding the template view to the native ad asset</p> <p>Recommended minimum width and height for the Star Rating UIImageView is to follow aspect ratio of W/H=5 such as <b>100x20</b>.</p>
Likes	No	UILabel	<p>Number of social ratings or “likes” of the product being offered to the user.</p> <p><b>Parameters:</b> Maximum likes length. The Likes UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>20 characters</b>.</p>
Downloads	No	UILabel	<p>Number of downloads of the product being offered to the user.</p> <p><b>Parameters:</b> Maximum downloads length. The Downloads UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p>



			It is recommended that the UILabel is laid out to display at least <b>25 characters</b> .
Price	No	UILabel	<p>Price for product / app / in-app purchase. Value should include currency symbol in localized format.</p> <p><b>Parameters:</b> Maximum Price length. The Price UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>20 characters</b>.</p>
Sale price	No	UILabel	<p>Sale price that can be used together with price to indicate a discounted price compared to a regular price. Value should include currency symbol in localized format.</p> <p><b>Parameters:</b> Maximum Sale price length. The Sale price UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>15 characters</b>.</p>
Phone	No	UILabel	<p>Formatted string that represents the phone number.</p> <p><b>Parameters:</b> Maximum phone length. The Phone UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>20 characters</b>.</p>
Address	No	UILabel	<p>Address data.</p> <p><b>Parameters:</b> Maximum Address length. The Address UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>100 characters</b>.</p>
Description 2	No	UILabel	Additional descriptive text associated with the



product or service being advertised.

**Parameters:** Maximum Address 2 length. The Address 2 UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.

It is recommended that the UILabel is laid out to display at least **100 characters**.

Display URL	No	UILabel	<p>Display URL data.</p> <p><b>Parameters:</b> Maximum display URL length. The Display URL UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>50 characters</b>.</p>
Call to action "CTA"	No	UIButton	<p>Descriptive text describing a 'call to action' button for the destination URL such as open, register, install.</p> <p><b>Parameters:</b> Maximum CTA length. The CTA UIButton is used to infer the maximum label length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the label of button is laid out to display at least <b>20 characters</b>.</p>
Views	No	UILabel	<p>Number of times the ad has been viewed.</p> <p><b>Parameters:</b> Maximum Views length. The Views UILabel is used to infer the maximum title length, unless it is explicitly set while binding the template view to the native ad asset.</p> <p>It is recommended that the UILabel is laid out to display at least <b>20 characters</b>.</p>

### Step 3.2 Adopting ADFNativeAdTemplate Protocol

ADFNativeAdTemplate protocol is defined as:

```
@class ADFNativeAdBinder;
@protocol ADFNativeAdTemplate <NSObject>

@required
-(void) bindViews:(ADFNativeAdBinder*) binder;
-(CGSize) sizeForWidth:(CGFloat) width;

@optional
```



```
-(void) renderExtraData:(NSMutableArray*) extraData;
@end
```

Below is a description of ADFNativeAdTemplate protocol's required methods:

- **bindViews: (ADFNativeAdBinder\*) binder:**  
This method is used to connect your native ad template's views to their corresponding native ad assets (title view, main image, icon, etc.).

You should at minimum bind one of the mandatory native ad assets (icon, title or MainAsset) depending on the native ad template design in-place.

Optionally, the binder allows setting additional parameters of the native ad assets such as the width and height of the icon and image, the maximum length of the title otherwise the binder will infer this information from the UIView's properties.

Binding views to native ad assets is achieved using the binder object of [ADFNativeAdBinder](#) class by calling the appropriate binding method (setIconImageView, setTitleLabel, setMainAssetView and setExtraDataView).

It is important to note that optional native assets are bound using the setExtraData method where each asset is represented by a Data ID such as:

- **ADF\_DATA\_ID\_DESC** for native ad description
- **ADF\_DATA\_ID\_CTA** for native ad click-through action label (install, open...etc.)
- **ADF\_DATA\_ID\_RATING** for native ad product rating

Please refer to Appendix 6 [ADFNativeAdBinder](#) for more details.

Below is a sample bindViews method implementation:

```
-(void) bindViews:(ADFNativeAdBinder*) binder
{
    [binder setIconImageView:self.iconImageView];
    [binder setAdChoicesView:self.daaView];
    [binder setTitleLabel:self.titleLabel];
    [binder setMainAssetView:self.adImage];
    [binder setExtraDataView:self.sponsoredLabel dataID:ADF_DATA_ID_SPONSORED];
    [binder setExtraDataView:self.descriptionLabel dataID:ADF_DATA_ID_DESC];
    [binder setExtraDataView:self.ratingImage dataID: ADF_DATA_ID_RATING];
    [binder setExtraDataView:self.actionButton dataID: ADF_DATA_ID_CTA];
}
```

- **sizeForWidth: (CGFloat) width:**  
This method is used to calculate the size of your native ad template view for the given parent view width or screen width. This is used while inferring the different attributes of your native ad views such as maximum length of your text views, and width and height of your image views.

```
-(CGSize)sizeForWidth:(CGFloat)width
{
    return CGSizeMake(width, TEMPLATE_AD_HEIGHT);
}
```



## Step 4: Requesting a Native Ad

1. Add `ADFNativeAd` to the view controller header file
  1. Import **AdFalconSDK** in your .h file .
  2. Declare **ADFNativeAd** instance.

Below is an example of how the header should look like:

```
@import "AdFalconSDK";

@interface AdFalcon_iOS_SDK_iPhone_SampleViewController: UIViewController{
    ADFNativeAd * nativeAd;
}
```

2. Add the following code snippet to `viewDidLoad` method

```
nativeAd = [[ADFNativeAd alloc] init];

//Enable or disable testing mode
nativeAd.testing = NO;

//Initialize native ad and pass your site ID, your template and view controller
[nativeAd
    initWithSiteID:@"Your Site ID"
    adTemplateClass:[your_ad_template class]
    viewController:self
];

//Add AdFalcon view
[self.view addSubview:nativeAd];

//Load ad from the AdFalcon service
[nativeAd loadAdWithParams:nil];
```

**Note 1:** It is highly recommended that you provide all available targeting information such as gender, location, content category to your ad request in order to receive the best ad that meets the criteria of your audience and maximize your return. This can be done by filling the **params** object in **loadAdWithParams** method. Please refer to Appendix 6 [ADFTargetingParams](#) for all the available targeting parameters.

**Note 2:** In case you are overriding the `[UIViewController loadView]` to programmatically draw your view, ensure that AdFalcon view is initialized after completing loading the `UIViewController's` view.

**Note 3:** Ensure testing parameter is set to No before uploading your app to Apple store.



## Step5: (Optional) Tracking Ad Lifecycle Events – ADFNativeAdDelegate Protocol

AdFalcon iOS SDK provides ability to track Ad lifecycle events. Follow the procedure below to subscribe to the various Ad lifecycle events

- Import the ADFADViewDelegate.h protocol to your header as the below:

```
@interface viewController: UIViewController<ADFNativeAdDelegate> {
```

- Add the below ADFADViewDelegate's methods to your source file as the below

```
-(void) nativeAdDidLoad:(ADFNativeAd*) nativeAd
{
    //Do something here
}

-(void) nativeAd:(ADFNativeAd*) nativeAd didFailWithErrorCode:(int) code
message:(NSString*) message
{
    //Do something here
}
```

Assign your delegate object to ADFAdView object as the below:

```
nativeAd.delegate = self;
```



## 5 Adding Native Ad to UITableView

AdFalcon SDK provides a helper class that enables you to easily add native ad to your table view while abstracting the complexity of working with UITableView.

### Step 1: Add ADFHelpers classes to your project.

Add the table helper classes in “ADFHelpers” folder in the example app project which is included in the SDK package to your project.

### Step 2: Open Your ViewController’s header file.

1. Import **ADFTableViewAdFiller.h**, **UITableView+ADFTableViewAdFiller.h**
2. Import your native ad template

Below is an example of how the header should look like after completing step 2:

```
#import "ADFTableViewAdFiller.h"
#import "ADFTableView+ADFTableViewAdFiller.h"
#import "YourTemplate.h"
```

### Step 3: Create Your Template View

In this step, you create your native ad template view, this template must extend **UIView** and adopt **ADFNativeAdTemplate** Protocol.

For the instructions to create your template view refer to [Creating Native Ad Template View Section](#)

### Step 4: Open Your ViewController’s Implementation file.

Add the below code to the **viewDidLoad** method, make sure it is added after you finish loading your data and set your datasource and delegate to your table view.

```
[ADFTableViewAdFiller
fillNativeAdWithSiteId:@"SITE ID that issued by AdFalcon"
numberOfAds:3 //Maximum number of Ads
startingPos:5 //The index of first AD
space:20 //Spaces between each Ad
adTemplateClass:[AppWallTemplate class] //Your template class
viewController:self // Your view controller
view:self.tableView //The table view you want your ads placed in
startHidden: NO //No means the cell of the ad will be visible while the ad is loading,
YES means the cell will be hidden until the ad is loaded.];
```

**Note:** Update the arguments passed to the `[ADFTableViewAdFiller fillNativeAdWithSiteId]` to values that meets your requirements.





## Step 5: Replace UITableView's methods

You must prefix all UITableView's methods with "adf\_". In case you missed any method, your table view will not be drawn correctly and will not function properly.

Instead of:

```
[self.tableView reloadData];
```

Use:

```
[self.tableView adf_reloadData];
```

The table below includes a full list of the classes and methods that you will need to rename to their AdFalcon equivalents:

Original Name	New Name
- (void) beginUpdate;	- (void) adf_beginUpdate;
- (void) endUpdate;	- (void) adf_endUpdate;
- (void) reloadData;	- (void) adf_reloadData;
- (void) deleteRowsAtIndexPaths:(NSArray *)indexPaths withRowAnimation:(UITableViewRowAnimation)animation;	- (void) adf_deleteRowsAtIndexPaths:(NSArray *)indexPaths withRowAnimation:(UITableViewRowAnimation)animation;
- (void) moveRowAtIndexPath:(NSIndexPath *)indexPath toIndexPath:(NSIndexPath *)newIndexPath;	- (void) adf_moveRowAtIndexPath:(NSIndexPath *)indexPath toIndexPath:(NSIndexPath *)newIndexPath;
- (void) reloadRowsAtIndexPaths:(NSArray *)indexPaths withRowAnimation:(UITableViewRowAnimation)animation;	- (void) adf_reloadRowsAtIndexPaths:(NSArray *)indexPaths withRowAnimation:(UITableViewRowAnimation)animation;
- (void) insertRowsAtIndexPaths:(NSArray *)devIndexPaths withRowAnimation:(UITableViewRowAnimation)animation;	- (void) adf_insertRowsAtIndexPaths:(NSArray *)devIndexPaths withRowAnimation:(UITableViewRowAnimation)animation;
- (void) moveSection:(NSInteger)section toSection:(NSInteger)newSection;	- (void) adf_moveSection:(NSInteger)section toSection:(NSInteger)newSection;
- (void) insertSections:(NSIndexSet *)sections withRowAnimation:(UITableViewRowAnimation)animation;	- (void) adf_insertSections:(NSIndexSet *)sections withRowAnimation:(UITableViewRowAnimation)animation;
- (void) deleteSections:(NSIndexSet *)sections withRowAnimation:(UITableViewRowAnimation)animation;	- (void) adf_deleteSections:(NSIndexSet *)sections withRowAnimation:(UITableViewRowAnimation)animation;
- (void) reloadSections:(NSIndexSet *)sections withRowAnimation:(UITableViewRowAnimation)animation;	- (void) adf_reloadSections:(NSIndexSet *)sections withRowAnimation:(UITableViewRowAnimation)animation;
- (void) selectRowAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated scrollPosition:(UITableViewScrollPosition)scrollPosition;	- (void) adf_selectRowAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated scrollPosition:(UITableViewScrollPosition)scrollPosition;
- (NSInteger)numberOfSections;	- (NSInteger)adf_numberOfSections;



- (NSInteger)numberOfRowsInSection:(NSInteger)section;	- (NSInteger)adf_numberOfRowsInSection:(NSInteger)section;
- (CGRect)rectForRowAtIndexPath:(NSIndexPath *)indexPath;	- (CGRect)adf_rectForRowAtIndexPath:(NSIndexPath *)indexPath;
- (NSIndexPath *)indexPathForRowAtPoint:(CGPoint)point;	- (NSIndexPath *)adf_indexPathForRowAtPoint:(CGPoint)point;
- (NSIndexPath *)indexPathForCell:(UITableViewCell *)cell;	- (NSIndexPath *)adf_indexPathForCell:(UITableViewCell *)cell;
- (NSArray *)indexPathsForRowsInRect:(CGRect)rect;	- (NSArray *)adf_indexPathsForRowsInRect:(CGRect)rect;
- (UITableViewCell *)cellForRowAtIndexPath:(NSIndexPath *)indexPath;	- (UITableViewCell *)adf_cellForRowAtIndexPath:(NSIndexPath *)indexPath;
- (NSArray *)visibleCells;	- (NSArray *)adf_visibleCells;
- (NSArray *)indexPathsForVisibleRows;	- (NSArray *)adf_indexPathsForVisibleRows;
- (void)scrollToRowAtIndexPath:(NSIndexPath *)indexPath atScrollPosition:(UITableViewScrollPosition)scrollPosition animated:(BOOL)animated;	- (void)adf_scrollToRowAtIndexPath:(NSIndexPath *)indexPath atScrollPosition:(UITableViewScrollPosition)scrollPosition animated:(BOOL)animated;
- (NSIndexPath *)indexPathForSelectedRow;	- (NSIndexPath *)adf_indexPathForSelectedRow;
- (NSArray *)indexPathsForSelectedRows	- (NSArray *)adf_indexPathsForSelectedRows

**Note:** if you override ***editActionsForRowAtIndexPath*** delegate's method of ***UITableViewDelegate***, you must return an array of ***ADFTableViewRowAction*** instead of ***UITableViewRowAction***.



## 5 Adding Native Ad to UICollectionView

AdFalcon SDK provides a helper class that enables you to easily add native ad to your UICollectionView while abstracting the complexity of working with UICollectionView.

### Step 1: Add ADFHelpers classes to your project.

Add the collection helper classes in “ADFHelpers” folder in the example app project which is included in the SDK package to your project.

### Step 2: Open Your ViewController’s header file.

3. Import **ADFCollectionViewFiller.h**, **UICollectionView+ADFCollectionViewFiller.h**
4. Import your native ad template

Below is an example of how the header should look like after completing step 2:

```
#import "ADFCollectionViewFiller.h"
#import "UICollectionView+ADFCollectionViewFiller.h"
#import "YourTemplate.h"
```

### Step 3: Create Your Template View

In this step, you create your native ad template view, this template must extend **UIView** and adopt **ADFNativeAdTemplate** Protocol.

For the instructions to create your template view refer to [Creating Native Ad Template View Section](#)

### Step 4: Open Your ViewController’s Implementation file.

Add the below code to the **viewDidLoad** method, make sure it is added after you finish loading your data and remember to set your datasource and delegate to your collection view.

```
[ADFCollectionViewFiller
 fillNativeAdWithSiteId:@"SITE ID that issued by AdFalcon"
 numberOfAds:3 //Maximum number of Ads
 startingPos:5 //The index of the first AD
 space:20 //Spaces between each Ad
 adTemplateClass:[YourTemplate class] //Your template class
 viewController:self // Your view controller
 view:self.collectionView //The collection view you want your ads placed in
 startHidden:NO //setting this to NO will cause the cell of the ad to be visible while the
 ad is loading, setting it to YES will cause the cell to be hidden until the ad is fully
 loaded.];
```

**Note:** Update the arguments passed to the `[ADFCollectionViewFiller fillNativeAdWithSiteId]` to values that meets your requirements.

### Step 5: Replace UICollectionView’s methods

You must prefix all UICollectionView’s methods with “adf\_”. In case you missed any method, your collection view will not be drawn correctly and you might face multiple issues in your collection view.



Instead of:

```
[self.collectionView reloadData];
```

Use:

```
[self.collectionView adf_reloadData];
```

Check the below table to ensure that you do not miss any method or class

Original Name	New Name
- (void) reloadData;	- (void) adf_reloadData;
- (void) deleteItemsAtIndexPaths:(NSArray *)indexPaths;	- (void) adf_deleteItemsAtIndexPaths:(NSArray *)indexPaths
- (void) moveItemAtIndexPath:(NSIndexPath *)indexPath toIndexPath:(NSIndexPath *)newIndexPath;	- (void) adf_moveItemAtIndexPath:(NSIndexPath *)indexPath toIndexPath:(NSIndexPath *)newIndexPath;
- (void) insertItemsAtIndexPaths:(NSArray *)devIndexPaths;	- (void) adf_insertItemsAtIndexPaths:(NSArray *)devIndexPaths;
- (void) moveSection:(NSInteger)section toSection:(NSInteger)newSection;	- (void) adf_moveSection:(NSInteger)section toSection:(NSInteger)newSection;
- (void) insertSections:(NSIndexPath *)sections;	- (void) adf_insertSections:(NSIndexPath *)sections;
- (void) deleteSections:(NSIndexPath *)sections;	- (void) adf_deleteSections:(NSIndexPath *)sections;
- (void) reloadSections:(NSIndexPath *)sections;	- (void) adf_reloadSections:(NSIndexPath *)sections;
- (void) selectItemAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated scrollPosition:(UICollectionViewScrollPosition)scrollPosition;	- (void) adf_selectItemAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated scrollPosition:(UICollectionViewScrollPosition)scrollPosition;
- (NSInteger)numberOfSections;	- (NSInteger)adf_numberOfSections;
- (NSInteger)numberOfItemsInSection:(NSInteger)section;	- (NSInteger)adf_numberOfItemsInSection:(NSInteger)section;
- (NSIndexPath *)indexPathForItemAtPoint:(CGPoint)point;	- (NSIndexPath *)adf_indexPathForItemAtPoint:(CGPoint)point;
- (NSIndexPath *)indexPathForCell:(UICollectionViewCell *)cell;	- (NSIndexPath *)adf_indexPathForCell:(UICollectionViewCell *)cell;
- (NSArray *)indexPathForItemAtPoint:(CGPoint)point;	- (NSArray *)adf_indexPathForItemAtPoint:(CGPoint)point;
- (UICollectionViewCell *)cellForItemAtIndexPath:(NSIndexPath *)indexPath;	- (UICollectionViewCell *)adf_cellForItemAtIndexPath:(NSIndexPath *)indexPath;
- (NSArray *)visibleCells;	- (NSArray *)adf_visibleCells;
- (NSArray *)indexPathsForVisibleItems;	- (NSArray *)adf_indexPathsForVisibleItems;
- (void)scrollToItemAtIndexPath:(NSIndexPath *)indexPath atScrollPosition:(UICollectionViewScrollPosition)scrollPosition animated:(BOOL)animated;	- (void)adf_scrollToItemAtIndexPath:(NSIndexPath *)indexPath atScrollPosition:(UICollectionViewScrollPosition)scrollPosition animated:(BOOL)animated;
- (NSArray *)indexPathsForSelectedItems;	- (NSArray *)adf_indexPathsForSelectedItems;



- (id)dequeueResuableCellWithReuseIdentifier:(NSString *) identifier forIndexPath:(NSIndexPath *) indexPath;	- (id)adf_dequeueResuableCellWithReuseIdentifier:(NSString *) identifier forIndexPath:(NSIndexPath *) indexPath;
-(void)reloadItemsAtIndexPaths:(NSArray *) indexPaths;	-(void)adf_reloadItemsAtIndexPaths:(NSArray *) indexPaths
-(void)deselectItemAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated;	-(void)adf_deselectItemAtIndexPath:(NSIndexPath *)indexPath animated:(BOOL)animated;
-(UICollectionViewLayoutAttributes *) layoutAttributesForItemAtIndexPath:(NSIndexPath *) indexPath;	-(UICollectionViewLayoutAttributes *) adf_layoutAttributesForItemAtIndexPath:(NSIndexPath *) indexPath;
-(UICollectionViewLayoutAttributes *) layoutAttributesForSupplementaryElementOfKind:(NSString *) kind atIndexPath:(NSIndexPath *) indexPath;	-(UICollectionViewLayoutAttributes *) adf_layoutAttributesForSupplementaryElementOfKind:(NSString *) kind atIndexPath:(NSIndexPath *) indexPath;
-(void)performBatchUpdates:(void (^)(void))updates completion:(void (^)(BOOL finished))completion;	-(void)adf_performBatchUpdates:(void (^)(void))updates completion:(void (^)(BOOL finished))completion;



## 6 Appendix

### ADFUserInfo Class

The ADFUserInfo Class sets the information of app's user.

The properties of the ADFUserInfo Class are:

Parameter	Required	Description	Values
Language	No	The language of the requested ad in ISO 639-1 language codes format (Two Letters code);	ar, en
Postal code	No	The user's postal/ZIP code	11121
Area code	No	The user's area code	06
Age	No	The user's age	27
Gender	No	The user's gender	kADFUserInfoGenderNone kADFUserInfoGenderMale kADFUserInfoGenderFemale
Country code	No	County code of the end user in ISO 3166-1 alpha-2 format code (two-letter code)	JO, SA ...etc.
Birthdate	No	Birthdate of app user in format dd.MM.yyyy	21.11.1984
Location: Latitude, Longitude	No	The geolocation information of the device. The location information is divided into two double values; latitude and longitude.	35.658, 34.641



## ADFTargetingParams Class

The ADFTargetingParams class sets the user and app information in order to help AdFalcon to deliver the most related and targeted ad to the user. All properties are optional.

The properties of the ADFTargetingParams Class are:

Parameter	Required	Description	Values
userInfo	No	A class containing information about the user of app	<a href="#">ADFUserInfo</a>
Keywords	No	A list containing keywords in comma separated format. AdFalcon's ads selector engine will search for ads relevant these keywords.	ex. sport, news, lifestyle, etc.
Additional Info	No	A map of keys and values to add additional parameters.	

## ADFNativeAd Class

Property	Attribute	Description
estimatedSize	Read only	Returns expected size based on the passing ad template in <b>initialize</b> method
status	Read only	Returns enum value of ADFNativeAdStatus type, this is used to know the current status for native ad. The possible values are the following: <ul style="list-style-type: none"> <li>• kADFNativeAdStatusNone</li> <li>• kADFNativeAdStatusLoading</li> <li>• kADFNativeAdStatusReady</li> <li>• kADFNativeAdStatusFailed</li> <li>• kADFNativeAdStatusClicked</li> </ul>
ready	Read only	Boolean value, returns yes if the ad is loaded successfully and ready to show. No if the ad is still loading, failed or not loaded yet.



errorMessage	Read only	String value, this value will be filled, if the status of native ad is failed.
testing	Assign	The property is used to inform AdFalcon network that the app is under the testing mode rather than production mode. Before releasing your app ensure this is set to NO.
Logging	Assign	Enable logging
delegate	Weak	ADFNativeAdDelegate value, this property is used to pass the delegate of native ad.

Method	Description
- (void) initializeWithSiteID:(NSString*) siteID adTemplateClass:(Class) adTemplateClass viewController:(UIViewController*) viewController;	This method should be called only once, before calling loadAd method. <b>Method Parameters:</b> <ul style="list-style-type: none"> <li>siteID: pass the site id which you got from adfalcon website.</li> <li>adTemplate: pass your ad template class.</li> <li>viewController: pass the view controller which contains the native ad.</li> </ul>
- (void) loadAdWithParams:(ADFTargetingParams*) targetingParams;	This method is used to load new ad. <b>Method Parameters:</b> <ul style="list-style-type: none"> <li>ADFTargetingParams: Ad Request Targeting Parameters</li> </ul>

## ADFNativeAdTemplate Protocol

Method	Required	Description
-(void) bindViews:(ADFNativeAdBinder*) binder;	Yes	<p>This method is used to connect your native ad template's views to their corresponding native ad assets (title view, main image, icon, etc.).</p> <p>You should at minimum bind one of the mandatory native ad assets (icon, title or MainAsset) depending on the native ad template design in-place.</p> <p>Optionally, the binder allows setting additional parameters of the native ad assets such as the width and height of the icon and image, the maximum length of the title otherwise the binder will infer this information from the UIView's properties.</p> <p>Binding views to native ad assets is achieved using the binder object of <a href="#">ADFNativeAdBinder</a> class by calling the appropriate binding method (setIconImageView, setTitleLabel, setMainAssetView and setExtraData).</p>





<code>-(CGSize) sizeForWidth:(CGFloat) width;</code>	Yes	This method is used to calculate the size of your native ad template view for the given parent view width or screen width. This is used while inferring the different attributes of your native ad views such as maximum length of your text views, and width and height of your image views.
<code>-(void) renderExtraData:(NSDictionary*) extraData</code>	No	This method is used to allow the app to draw any extra data assets that were not bound explicitly in <b>bindViews</b> method. This is used if the app is willing to entirely take over the extra data display and presentation.

## ADFNativeAdDelegate Protocol

Method	Required	Description
<code>-(void) nativeAdDidLoad:(ADFNativeAd*) nativeAd;</code>	Yes	Fired when the ad is loaded successfully.
<code>-(void) nativeAd:(ADFNativeAd*) nativeAd didFailWithErrorCode:(int) code message:(NSString*) message;</code>	Yes	Fired when failed to load an ad with failure reason.
<code>-(BOOL) nativeAd:(ADFNativeAd*) nativeAd handleCustomActionWithData:(NSString*) customData;</code>	No	<p>This method will be invoked by the SDK if the action type of an ad is custom.</p> <p><b>Method Parameters:</b></p> <ul style="list-style-type: none"> <li>ADFNativeAd: Native ad object</li> <li>customData: The Ad Custom data object in the format agreed on between the publisher and the advertiser; normally this is in JSON format.</li> </ul> <p><b>Return Value:</b> Returns YES if the action has been handled successfully, otherwise return NO.</p>
<code>-(void) nativeAdWillPresentScreen:(ADFNativeAd*) nativeAd;</code>	No	This method will be invoked before presenting the ad screen.
<code>-(void) nativeAdDidPresentScreen:(ADFNativeAd*) nativeAd;</code>	No	This method will be invoked after presenting the ad screen.
<code>-(void) nativeAdWillDismissScreen:(ADFNativeAd*) nativeAd;</code>	No	This method will be invoked before dismissing the ad screen.



-(void) nativeAdDidDismissScreen:(ADFNativeAd*) nativeAd;	No	This method will be invoked after dismissing the ad screen.
---	----	---

## ADFNativeAdError Enum

Enum	Description
kADFNativeAdErrorInternalServer	is an error that's happened within the web server attempting to get you an ad. It's typically a server-side problem out of your control
kADFNativeAdErrorNoAdAvailabe	no ad available in AdFalcon stores
kADFNativeAdErrorInvalidParam	There is a parameter has invalid value
kADFNativeAdErrorMissingParam	You have missed to fill required parameter
kADFNativeAdErrorGenericSDK	An error happened within the SDK during attempting to load or render an ad.
kADFNativeAdErrorCommunication	No connection available to the internet.
kADFNativeAdErrorInterstitialAlreadyUsed	An error happened when application try to reuse Interstitial

## ADFNativeAdBinder Class

Method	Description
-(void) setIconImageView:(UIImageView*) imageView;	Binds the native ad icon asset to its corresponding UIImageView in the template.
-(void) setIconImageView:(UIImageView*) imageView minSize:(CGSize) minSize;	<p>The first method is used to pass the template's icon image view. The SDK will determine the image view size and return the best size for your icon.</p> <p>Or you can use the overloaded method to explicitly set the minimum size of the requested icon.</p> <p>The size of any icon must be a square i.e. the width is equal to the height.</p> <p>Recommended minimum width and height of the Icon UIImageView is <b>50x50</b>.</p>



```
-(void) setTitleLabel:(UILabel*) label;
```

Binds the native ad title asset to its corresponding **UILabel** in the template.

```
-(void) setTitleLabel:(UILabel *)label  
maxLength:(CGFloat) maxLength;
```

The first method is used to pass the label view of the template's title. The SDK will infer the maximum length of the label based on **UILabel** properties.

Or you can use the overloaded method to explicitly set the maximum length of the title.

It is recommended that the UILabel is laid out to display at least **25 characters**.

```
-(void) setMainAssetView:(UIView *)  
view;
```

Binds the native ad main asset to its corresponding **UIView** in the template.

```
-(void) setMainAssetView:(UIView *)  
view minSize:(CGSize) minSize;
```

The MainAsset is a UIView which is used as a container view for the actual native ad main asset which can be:

- Image, or
- XHTML (HTML+JavaScript), or
- Video

The SDK will automatically create the adequate UIView type that match the native ad main asset.

The first method is used to pass the main asset's view, the SDK will use it to infer the minimum adequate width and height.

Or you can use the overloaded method to explicitly set the minimum required size for the main asset.

The main asset UIView width and height aspect ratio should always be W/H=1.91. It is recommended to use 320x167 points.

```
-(void) setExtraDataView:(UIView *)  
view dataID:(NSInteger) dataID;
```

Binds the native ad data asset to its corresponding **UIView** in the template.

```
-(void) setExtraDataView:(UIView *)  
view dataID:(NSInteger) dataID  
maxLength:(CGFloat) maxLength;
```

All extra data views are optional which means native ad can be returned even if they don't include the requested extra data assets.

```
-(void) setExtraDataLabel:(UILabel *)  
label dataID:(NSInteger) dataID  
maxLength:(CGFloat) maxLength
```

Additionally, the SDK will hide any extra data view that is not included in the returned native ad.

```
-(void)  
setExtraDataImageView:(UIImageView *)  
imageView dataID:(NSInteger) dataID  
minSize:(CGSize) minSize
```

The first method is used to pass the view with its data id, the SDK will use the view properties to infer any needed additional parameters, such text max length, or images height and width.



The overloaded method is used to explicitly set any additional parameters.

Below is a list of supported data asset types of native ads:

Data	ID	View
ADF_DATA_ID_SPONSORED	1	UILabel
ADF_DATA_ID_DESC	2	UILabel
ADF_DATA_ID_RATING	3	UIImageView
ADF_DATA_ID_LIKES	4	UILabel
ADF_DATA_ID_DOWNLOADS	5	UILabel
ADF_DATA_ID_PRICE	6	UILabel
ADF_DATA_ID_SALEPRICE	7	UILabel
ADF_DATA_ID_PHONE	8	UILabel
ADF_DATA_ID_ADDRESS	9	UILabel
ADF_DATA_ID_DESC2	10	UILabel
ADF_DATA_ID_DISPLAYURL	11	UILabel
ADF_DATA_ID_CTA	12	UIButton
ADF_DATA_ID_VIEWS	13	UILabel

starRatingEmptyColor

Change the color of empty area in star rating

starRatingFillingColor

Change the color of filling area in star rating

-(void) setAdChoicesView:(UIView \*)  
view;

Binds the native ad AdChoices asset to its corresponding **UIView** in the template. It is a **Mandatory** field.

An AdChoices icon must be displayed in your ad to enable the user to learn about interest-based advertising.

The SDK will decide when it is needed to display the icon and will also handle the tap event on the view by opening an opt-out page in the browser.

The size of the view: 20x20.

You must place the view in any one of the four corners of the ad template; upper right hand corner is recommended.



## 7 More Information:

You can find more information in the example project.

For any SDK integration queries, please send us an email to [support@adfalcon.com](mailto:support@adfalcon.com) along with your login id.