iOS SDK Documentation

Version 1.1

135 Tech Labs

Contents

Chapter 1

AceSDK

The iOS version of the AceSDK provides an independent and uniform interface for iOS applications to interact with the Hangoutt reward platform.

Through the use of the native AceViewController, the AceSDK intelligently implements the complete rewarding cycle: score reporting, fetching and displaying of rewards in a separate overlaid view and tracking the reward redemption.

AceSDK also features serendipitous rewarding - a way of identifying rewardable instances for the user in the application lifecycle, and rewarding the user at these instances. This is explained in detail in Section 1.6.

1.1 Overview

1.1.1 How to use AceSDK?

1. Drag the AceSDK.framework and the AceSDKResources.bundle into the project. (Section 1.2)
2. Import the AceSDK.h in AppDelegate.h and application files where the functions of AceSDK are called. (Section 1.2.1)
3. Report the score of the user. (Section 1.3)
4. Reward the user. (Sections 1.5 and 1.6)
5. (Optional) Get the user's score. (Section 1.4)

1.1.2 Where to use AceSDK?

1. Could be used when the user completes a high level score in a game – the highest emotional quotient.
2. At the time of achieving goals inside of apps (like completing a mile run in a fitness app, completes listening to top 5 songs of the day).
3. When the user shares an important feature of an app or content in the app.
4. When a user does an "in-app purchase".

1.2 Importing AceSDK

In order to import AceSDK into a third-party application, two files need to be added to the corresponding project. The AceSDK.framework file needs to be added to the frameworks section (build phases section) of the project. Also, the AceSDKResources.bundle file has to be added to the bundle resources section of the project.

Along with AceSDK.framework the following frameworks need to be added to the Build Phases section (if they are not already present):

- MobileCoreServices.framework
- Security.framework
- SystemConfiguration.framework
- Foundation.framework
- UIKit.framework

A linker flag has to be added to the Targets section of your application. Select the Build Settings tab and add the flag **-ObjC** to the option **Other Linker Flags**.

1.2.1 HangouttRewards Singleton

The AceSDK framework provides a single-point interface of invoking methods through the HangouttRewards singleton. In order to be able to access the HangouttRewards class, the following header file needs to be imported in the applica-

tion's AppDelegate.h file. In addition, the AceSDK.h file needs to be imported in every application file where methods on HangouttRewards are invoked.

```
#import <AceSDK/AceSDK.h>
```

The Hangoutt reward platform uses OAuth2 for authentication and authorization. When you sign up as a third-party developer with the Hangoutt platform, you will get a set of client credentials for each application that you add to the Hangoutt platform.

Each application receives as part of the client credentials, a `client_id` and a `client_secret`. These have to be specified as parameters when initializing HangouttRewards, for it to be able to successfully connect and interact with the Hangoutt reward platform.

We recommend that the HangouttRewards singleton be initialized in the `didFinishLaunchingWithOptions` method of the AppDelegate. This is illustrated in Figure 1.1.

```
- (BOOL)application:(UIApplication *)application
        didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    HangouttRewards *hangouttRewards = [[HangouttRewards alloc]
                                        initWithClientID:@"your_client_id"
                                        andClientSecret:@"your_client_secret"
                                        andGrantType:@"client_credentials"];

    [HangouttRewards setSharedInstance:hangouttRewards];
    return YES;
}
```

Figure 1.1: Initializing HangouttRewards.

Note: The application's `client_id` and `client_secret` should replace `your_client_id` and `your_client_secret`.

Invoking Methods on HangouttRewards

To invoke any method which is a part of HangouttRewards, the correct way is:

```
[[HangouttRewards sharedInstance] method_name];
```

1.3 Reporting the score

This method helps accumulate the score of the user over several invocations of the application. The score that is to be reported is passed as a parameter in string format. This method can be invoked in the following manner:

```
[[HangouttRewards sharedInstance] add:score];
```

The score that is reported is in terms of Hangoutt points earned. Please refer to Section 1.7 for a detailed explanation on how Hangoutt points are computed.

1.4 Querying the score

The application can request for the current score of the user as stored with AceSDK. The get method, when invoked, returns the score in string format.

```
NSString *score =
        [[HangouttRewards sharedInstance] get];
```

1.5 Rewards for the user

When the application needs AceSDK to fetch and display the rewards to the user it must invoke the `reward_user` method. This method starts the AceViewController which displays the rewards to the user in a separate view overlaid on the application's current UIViewController.

This method can be invoked as shown below, passing the invoking UIViewController as a parameter.

```
[[HangouttRewards sharedInstance] reward_user:self];
```

If this method is being called in a ViewController, then it should be called from the viewDidAppear method rather than the viewDidLoad method.

There is also another version of this method which allows for the specification of user segmentation parameters. This is explained in the next section.

1.6 Targeted rewards for the user

When the application needs AceSDK to fetch and display the rewards to the user, it must invoke this method. This method starts the AceViewController which displays the rewards to the user in a separate view overlaid on the application's UIViewController.

This method is invoked as shown below:

```
[[HangouttRewards sharedInstance] reward_user: self
        ifSerendipity: serendipity
        withParameters: parameters];
```

This method allows for an additional set of parameters to be passed which are used to control the categorization and ranking of rewards on the server side.

For applications that can specify a finer user profile for better reward targeting, the parameters dictionary (`NSMutableDictionary`) can be used to specify such properties. The set of parameters that can be sent are listed in Table 1.1.

This method also allows for the user to be rewarded serendipitously. This is a way of identifying rewardable instances for the user in the application lifecycle, and rewarding the user at these instances.

For fetching serendipitous rewards, the serendipity boolean needs to be set to true and the representative score needs to be sent as a parameter within the parameters dictionary (with "score" as key).

The parameters dictionary also allows for labels to be associated with each call, thereby allowing the developer to identify the reward moments. This is explained in greater detail in Section 1.8.

1.7 Hangoutt Points

Every third-party application has its own way of scoring, with scores not being directly comparable across applications in terms of either effort rendered or time spent. The Hangoutt reward platform, thus, had to define a universal scale of scoring so that consistency in number of points earned is maintained across applications.

We use a very simple formula to compute the user's score:

Key(String)	Value(String)
score	String representation of user's score. Only used when serendipity is set to true.
country_code	2 letter ISO defined Country Code (in uppercase)
country	Country Name (lowercase)
city	City Name (lowercase)
email	User email id
gender	Gender("male", "female" or "all")
limit	String representation of the maximum number of rewards to be returned. (Between 1-5)
title	The title that will be displayed to the user. (Maximum length – 60 characters with spaces)
age_from	String representation of the minimum age
age_to	String representation of the maximum age
labels	String value of JSONArray for tracking reward moments.

Table 1.1: Parameters that can be set for better reward targeting.

```
10 minutes of effort = 100 Hangoutt points.
```

In other words, we score based on the amount of time spent by a user actively engaging with the application. An active engagement duration of 10 minutes will fetch the user an average of 100 Hangoutt points. This is the benchmark that we follow in our applications, and recommend that every third-party application reporting scores to Hangoutt reward platform should follow the same benchmark.

1.8 Tracking effectiveness of reward moments

AceSDK allows the developer to track the redemption statistics and conversion rates at different reward moments within the application. This can be achieved by sending tracking information about the reward moments to the Hangoutt Rewards platform.

In a given third-party application, identifying which is an **appropriate** reward moment is the job of the developer. This is so because it is only the developer who has the complete information about usage patterns of the app to begin with. However, these reward moments must be monitored over time to validate the developer's intuition and also to achieve a more efficient monetization.

AceSDK enables this monitoring of reward moments through the use of **labels**. A label is any developer defined string that classifies a reward moment and also possibly distinguishes it from other rewarding moments. For example, a label can be:

- Source code location marker (e.g., View controller name, followed by function name where the `reward_user` function was called)
- Event marker (e.g., In-App purchase, Sharing the score on social networks, Achieving high score etc.)
- Total Engagement time (after which the `reward_user` method was called)
- Level completed (for games)

If we go through the above list, we realize that a given reward moment can be tagged with multiple labels. This is so because classifications of reward moments usually overlap. Thus having a single string label which concatenates **all** contextual information would, while achieving the goal of distinguishing a particular reward moment, defeat the ideal of grouping all similar reward moments together. For example, if the developer wants to track effectiveness of all reward moments associated with In-App purchases, it can only be done if the reward moment is tagged with multiple labels rather than a single unique label.

Therefore, AceSDK allows each call to `reward_user` to be tagged with multiple labels, which may or may not overlap with other calls to `reward_user` in the same application. The developer has to format all such labels as a string representation of a JSONArray, and provide this as the value for the key "labels" in the categorization parameters dictionary. We explain this below through an example as shown in Figure 1.2.

```
NSMutableDictionary *parameters;
BOOL hideUI = YES;
BOOL serendipity = NO;

-(void) rewardUser {
    NSArray *labels = @[@"EndViewController - rewardUser Method", @"Reason: In-App purchase"];

    NSData *data = [NSJSONSerialization dataWithJSONObject:labels
                                                   options:0 error:nil];

    NSString *labels_string = [[NSMutableString alloc]
                               initWithData:data encoding:NSUTF8StringEncoding];

    parameters = [[NSMutableDictionary alloc]init];
    [parameters setValue:@"IN" forKey:@"country_code"];
    [parameters setValue:@"india" forKey:@"country"];
    [parameters setValue:@"mumbai" forKey:@"city"];
    [parameters setValue:@"info@135techlabs.com" forKey:@"email"];
    [parameters setValue:@"25" forKey:@"age_from"];
    [parameters setValue:@"60" forKey:@"age_to"];
    [parameters setValue:@"3" forKey:@"limit"];
    [parameters setValue:labels_string forKey:@"labels"];

    [[HangouttRewards sharedInstance]
         reward_user:self
         requestRewardsCount:hideUI
         ifSerendipity:serendipity
         withParameters:parameters
         HangouttDelegate:self
    ];
}

-(void) rewardsCount:(NSString *)count {
    if(count > 0) {
        [[HangouttRewards sharedInstance]
             reward_user:self
             requestRewardsCount:!hideUI
             ifSerendipity:serendipity
             withParameters:parameters
             HangouttDelegate:self
        ];
    }
}
```

Figure 1.2: Sample invocation of the reward user method with tracking labels.

1.9 Querying for rewards before displaying

If a third-party application implements more than one SDK's for monetization, the control over UI cannot be seamlessly shared between such SDK's. The developer has to choose which SDK will be given preference in a given situation.

To enable this level of decision on the developer's side, we allow the third-party application to request only for the information on the rewards eligible for the user's score (or the score passed as a parameter in case of serendipitous rewarding). The decision on whether to display the rewards to the user can be made at a later time.

The third-party application wishing to implement this functionality has to invoke an instance of the `reward_user` method with a `count_only` boolean flag set to `true`. In addition, it also has to pass an instance of the HangouttDelegate which will be invoked when the reward count information becomes available. The

UIViewController implementing this functionality has to add an interface for the HangouttDelegate in the UIViewController.h file.

When `count_only` is set to `true`, AceSDK fetches the rewards and informs the third-party application about the number of eligible rewards through the HangouttDelegate. This functionality takes place in the background without displaying any overlaid view.

The third-party application can then decide whether to display the fetched rewards or not. The ideal place to do this is the HangouttDelegate's `rewardsCount` method. The same `reward_user` method can be called, but this time setting the `count_only` parameter to `false`. This informs AceSDK to display the previously fetched rewards to the user.

1.9.1 Rewards for the user

This is the query-version of the method explained in Section 1.5. The method can be invoked as follows:

```
[[HangouttRewards sharedInstance] reward_user:self
        requestRewardsCount: count_only
        HangouttDelegate:self];
```

1.9.2 Targeted rewards for the user

This is the query-version of the method explained in Section 1.6. The method can be invoked as follows:

```
[[HangouttRewards sharedInstance] reward_user:self
        requestRewardsCount: count_only
        ifSerendipity: serendipity
        withParameters: parameters
        HangouttDelegate:self];
```

A Sample invocation of the method is shown in Figure 1.3.

```
NSMutableDictionary *parameters;
BOOL hideUI = YES;
BOOL serendipity = NO;

-(void) rewardUser {
    [parameters setValue:@"IN" forKey:@"country_code"];
    [parameters setValue:@"india" forKey:@"country"];
    [parameters setValue:@"mumbai" forKey:@"city"];
    [parameters setValue:@"info@135techlabs.com" forKey:@"email"];
    [parameters setValue:@"25" forKey:@"age_from"];
    [parameters setValue:@"60" forKey:@"age_to"];
    [parameters setValue:@"3" forKey:@"limit"];

    [[HangouttRewards sharedInstance]
         reward_user:self
         requestRewardsCount:hideUI
         ifSerendipity:serendipity
         withParameters:parameters
         HangouttDelegate:self
    ];
}

-(void) rewardsCount:(NSString *)count {
    [[HangouttRewards sharedInstance]
         reward_user:self
         requestRewardsCount:!hideUI
         ifSerendipity:serendipity
         withParameters:parameters
         HangouttDelegate:self
    ];
}
```

Figure 1.3: Query for rewards.

Appendix A

Troubleshooting

1. The Application has to instantiate HangouttRewards before calling any method on it. This has to be done in the AppDelegate.m file as shown in Section 1.2.1.
2. Due to the changes in the UIViewController's life cycle in iOS 8+,
   - In iOS versions < iOS 8 when the AceViewController finishes its task and returns to the application's UIViewController that invoked the `reward_user` method, the `viewDidAppear` method is called on the UIViewController.
   - In iOS versions 8+ when the AceViewController finishes its task, the `viewDidAppear` method of the application's UIViewController is not called.
3. When the application requests for only the number of rewards, and then after being notified about the rewards, if it wishes to display the stored rewards, the parameters passed to the two requests must remain same. If the parameters differ, then the parameters that were passed during the call with `count_only` set to `true` are considered valid. Any change in the parameters will not be reflected in a subsequent call to `reward_user` with `count_only` set to `false`.