SDKLite Documentation

Version 1.0

135 Tech Labs

Contents

Chapter 1

AceSDKLite

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

Through the use of the native AceActivity (Section 1.2.2), 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.

1.1 Overview

1.1.1 How to use AceSDKLite?

1. Import AceSDKLite into the workspace (Section 1.2)
2. Connect your application to AceSDKLite (Section 1.3)
3. Reward the user (Section 1.4)

1.1.2 Where to use AceSDKLite?

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 AceSDKLite Library

The AceSDKLite library module has to be imported into the same workspace as the third-party application implementing the SDK. For more information on library modules, please refer to the relevant Android developer documentation for library modules.

In addition, the third-party application will have to add a reference to the AceSDKLite library module to access the shared code and resources. For more information on referencing library modules in Eclipse IDE, please refer to the relevant Android developer documentation for referencing libraries.

1.2.1 HangouttRewards enum

The AceSDKLite library module provides a single-point interface of invoking methods through the HangouttRewards enumeration. HangouttRewards is an implementation of the Singleton pattern through the enum data type, using which we provide the third-party developer a uniform interface to access AceSDKLite.

By itself, the enum cannot achieve creation and displaying of views on the Android platform. For this purpose, we use a single activity, called AceActivity, which is responsible for all views which form a part of AceSDKLite.

1.2.2 AceActivity

This activity is a part of AceSDKLite library module, which will be accessible to the developer if the AceSDKLite library module has been referenced correctly.

AceActivity overlays its views on top of the current activity, that is, the activity which called the corresponding SDK method. This is achieved through a custom style which is defined as part of the AceSDK library module.

AceActivity has to be declared as part of the third-party application to achieve our desired functionality. To do this, add the following activity declaration to your application’s manifest:

```
<activity
    android:name="com.techlabs135.ace.AceActivity"
    android:theme="@style/Theme.techlabs135.AceSDK"
    android:screenOrientation="portrait">
</activity>
```

1.2.3 Internet Permissions

As the Hangoutt reward platform is only accessible through the Internet, the third-party application implementing AceSDKLite has to request for the requisite internet permissions from the device. To do this, add the following uses-permission directives to your application's manifest:

```
<uses-permission
   android:name="android.permission.INTERNET"/>
<uses-permission
   android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

1.2.4 Client Credentials

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 part of the third party application resources, for it to be able to successfully connect and interact with the Hangoutt reward platform. In order to do this, first declare the `client_id` and `client_secret` as string resources in your application project's strings.xml:

```
<string name="client_id">your_client_id</string>
<string name="client_secret">your_client_secret</string>
<string name="grant_type">client_credentials</string>
```

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

Now, the developer has to refer to these string resources in the application manifest file as follows:

```
<meta-data
   android:name="com.techlabs135.ace_client_id"
   android:value="@string/client_id"/>
<meta-data
   android:name="com.techlabs135.ace_grant_type"
   android:value="@string/grant_type"/>
<meta-data
   android:name="com.techlabs135.ace_client_secret"
   android:value="@string/client_secret"/>
```

1.3 Connecting to the Hangoutt Reward Platform

To connect to the Hangoutt reward platform from any third-party application, the following method has to be invoked in the `onCreate` method of launcher activity of the third-party application:

```
HangouttRewards.INSTANCE.connect(this);
```

This method sets up the connection between the third-party application instance and the Hangoutt reward platform server. It will return true upon a connection being established successfully, false otherwise.

Note: This note is concerned with the method invocation syntax in AceSDKLite. As explained previously in Section 1.2.1, HangouttRewards is an enum implementation of the singleton pattern. To invoke any method which is a part of the HangouttRewards enum, the correct way is to refer to the INSTANCE value in the enum and invoke methods on it:

```
HangouttRewards.INSTANCE.<method_name>
```

1.4 Rewards for the user

When the application needs AceSDKLite to fetch and display the rewards to user it must invoke the `reward_user` method. This method starts the AceActivity

which displays the rewards to the user in a separate view overlaid on the application's current activity.

This method can be invoked as shown below, passing the invoking activity as a parameter. A sample invocation of the method is shown in Figure 1.1.

```
HangouttRewards.INSTANCE.reward_user(SomeActivity.this,
        count_only, parameters, listener);
```

```
public class SomeActivity extends Activity implements RewardsCountListener {
   private boolean hideUI=true;

   @Override
   public void onCreate(Bundle savedInstanceState) {
        // Some code
        HashMap<String,String> parameters = new HashMap<String, String>();
        parameters.put("score", "200");
        parameters.put("email", "info@135techlabs.com");
        parameters.put("country_code", "IN");
        parameters.put("country", "india");

        HangouttRewards.INSTANCE.reward_user(SomeActivity.this, !hideUI,
                        parameters, this);
   }

   @Override
   public void rewardsCount(int count) {
   }
}
```

Figure 1.1: Sample invocation of the reward user method.

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 HashMap can be used to specify such properties. The set of parameters that can be sent are listed in Table 1.1.

The parameters hashmap 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 1.6.

(Note: In the absence of an Internet connection, AceActivity will finish itself, without displaying the overlaid view, and without throwing any error.)

Key(String)	Value(String)
`score`	String representation of user's score.
`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.

1.5 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:

```
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 applica-

tion reporting scores to Hangoutt reward platform should follow the same benchmark.

1.6 Tracking effectiveness of reward moments

AceSDKLite 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.

AceSDKLite 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., Activity 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, AceSDKLite 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 hashmap. We explain this below through an example as shown in Figure 1.2.

```
public class SomeActivity extends Activity implements RewardsCountListener {
   private boolean hideUI=true;
   private HashMap<String,String> parameters = new HashMap<String, String>();

   @Override
   public void onCreate(Bundle savedInstanceState) {

       JSONArray labels = new JSONArray();
       labels.put(0, "SomeActivity - OnCreate Method");
       labels.put(1, "Reason:In-App purchase.");
       String labels_string = labels.toString();

       parameters.put("score", "250");
       parameters.put("country_code", "IN");
       parameters.put("country", "india");
       parameters.put("city", "mumbai");
       parameters.put("gender", "all");
       parameters.put("title", "You've achieved high score!! Let us reward you.");
       parameters.put("limit", "5");
       parameters.put("age_from", "17");
       parameters.put("age_to", "50");
       parameters.put("email", "info@135techlabs.com");

       parameters.put("labels", labels_string);

       HangouttRewards.INSTANCE.reward_user(SomeActivity.this, hideUI, parameters, this);
   }

   @Override
   public void rewardsCount(int count) {
       if (count > 0) {
          HangouttRewards.INSTANCE.reward_user(SomeActivity.this, !hideUI, parameters, this);
      }
   }
}
```

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

1.7 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. 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 RewardsCountListener which will be invoked when the reward count information becomes available.

When `count_only` is set to `true`, AceSDKLite fetches the rewards and informs the third-party application about the number of eligible rewards through the RewardsCountListener. 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 RewardsCountListener's `rewardsCount` method. The same `reward_user` method can be called, but this time setting the `count_only` parameter to `false`. This informs AceSDKLite to display the previously fetched rewards to the user.

A sample invocation of the `reward_user` method is shown in Figure 1.3.

```
public class SomeActivity extends Activity implements RewardsCountListener {
   private boolean hideUI=true;
   private HashMap<String,String> parameters =
                  new HashMap<String, String>();

   @Override
   public void onCreate(Bundle savedInstanceState) {
        // Some code
        parameters.put("score", "200");
        parameters.put("email", "info@135techlabs.com");
        parameters.put("country_code", "IN");
        parameters.put("country", "india");
        parameters.put("state", "maharashtra");
        parameters.put("city", "mumbai");
        parameters.put("gender", "all");
        parameters.put("limit", "5");
        parameters.put("age_from", "24");
        parameters.put("age_to", "50");

        HangouttRewards.INSTANCE.reward_user(SomeActivity.this, hideUI,
                       parameters, this);
    }

    @Override
    public void rewardsCount(int count) {
        if (count > 0) {
            HangouttRewards.INSTANCE.reward_user(SomeActivity.this, !hideUI,
                           parameters, this);
        }
    }
}
```

Figure 1.3: Sample invocation of the reward user method used with the parameters and the count only flag.

Appendix A

Troubleshooting

1. The `connect(Activity activity)` method has to be invoked by the application only on the `onCreate(Bundle savedState)` method of the launcher activity.
2. 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`.