SDK Documentation

Version 1.1

135 Tech Labs

Contents

List of Figures

List of Tables

Chapter 1

AceSDK

The Android version of the AceSDK 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.

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 1.7.

1.1 Overview

1.1.1 How to use AceSDK?

1. Import AceSDK into the workspace (Section 1.2)
2. Connect your application to AceSDK (Section 1.3)
3. Report the score of the user (Section 1.4)
4. Reward the user (Sections 1.6 and 1.7)
5. (Optional) Get the user's score (Section 1.5)

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 Library

The AceSDK 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 AceSDK 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.

(Note: If there is an error caused because of a difference in the versions of the library "android-support-v4.jar" in AceSDK and the application, please make sure that both the versions are same. Please copy the library from your application's Android Private Libraries into AceSDK's Android Private Libraries).

1.2.1 HangouttRewards enum

The AceSDK 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 AceSDK.

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 AceSDK.

1.2.2 AceActivity

This activity is a part of AceSDK library module, which will be accessible to the developer if the AceSDK 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 AceSDK 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 AceSDK. 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 Reporting the score

This method helps accumulate the score of the user over several invocations of the application. The parameters that need to be passed to this method are the activity which is invoking it, and the newly earned score that is to be reported. This method can be invoked in the following manner:

```
HangouttRewards.INSTANCE.add(SomeActivity.this, score);
```

The score that is reported is in terms of Hangoutt points earned. Please refer to Section 1.8 for a detailed explanation on how Hangoutt points are computed. For more information on this method, please refer to the SDK API Documentation.

1.5 Querying the current 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. The activity of the application invoking this method needs to be passed as a parameter.

```
String score =
        HangouttRewards.INSTANCE.get(SomeActivity.this);
```

1.6 Rewards for the user

When the application needs AceSDK 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.

```
HangouttRewards.INSTANCE.reward_user(SomeActivity.this);
```

An alternative method `reward_user`(Activity activity, Boolean serendipity, HashMap paramters) is also available. Please refer to Section 1.7 for more details.

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

1.7 Targeted Rewards for the user

When the application needs AceSDK to fetch and display rewards to the user, it must invoke this 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,
        serendipity, 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 HashMap can be used to specify such properties. The set of parameters that can be sent are listed in Table 1.1.

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

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.

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    //some code
    get_rewards.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View view) {
            HashMap<String, String> parameters = new HashMap<String, String>();
            parameters.put("email", "info@135techlabs.com");
            parameters.put("gender", "all");
            parameters.put("age_from", "15");
            parameters.put("age_to", "70");
            parameters.put("country_code", "IN");
            parameters.put("country", "india");
            parameters.put("state", "maharashtra");
            parameters.put("city", "mumbai");
            parameters.put("limit", "3");
            parameters.put("title", "Mission accomplished!!"+
               "Enjoy the moment with a reward.");

            HangouttRewards.INSTANCE.reward_user(EndActivity.this, false,
                parameters);
        }
    });
    //some code
}
```

Figure 1.1: Sample invocation of the reward user method.

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 HashMap (with “score” as key).

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 1.9.

1.8 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.

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.

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

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

```
public class EndActivity 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, "EndActivity - OnCreate Method");
       labels.put(1, "Reason:In-App purchase.");
       String labels_string = labels.toString();

       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(EndActivity.this, hideUI, parameters, this);
   }

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

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

1.10 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 stored 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 RewardsCountLis-

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

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 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 AceSDK to display the previously fetched rewards to the user.

1.10.1 Rewards for the user

The application can invoke the `reward_user` method without parameters (Section 1.6) with the above functionality in the following manner:

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

The `count_only` flag controls whether the UI is displayed to the user or not. A sample invocation of this method is shown in Figure 1.3.

```
public class EndActivity extends Activity implements RewardsCountListener {
     //some code
     boolean hideUI = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // some code
        HangouttRewards.INSTANCE.reward_user(EndActivity.this, hideUI, this);
        //some code
    }


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

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

1.10.2 Targeted Rewards for the user

The application can invoke `reward_user` method with parameters (Section 1.7) with the above functionality in the following manner:

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

The `count_only` flag controls whether the UI is displayed to the user or not. A sample invocation of this method is shown in Figure 1.4.

```
public class EndActivity extends Activity implements RewardsCountListener {
     //some code
     boolean hideUI = true;
     HashMap<String, String> parameters = new HashMap<String, String>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        // some code
        parameters.put("country_code", "IN");
        parameters.put("country", "india");
        parameters.put("city", "mumbai");
        parameters.put("gender", "all");
        parameters.put("title", "Reward title");
        parameters.put("age_from", "10");
        parameters.put("age_to", "100");
        parameters.put("email", "info@135techlabs.com");

        HangouttRewards.INSTANCE.reward_user(EndActivity.this, hideUI, false, parameters, this);
        //some code
    }


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

Figure 1.4: Sample invocation of the reward user method used with the parameters and the count only flag.

Chapter 2

Walkthrough with Linkem

Linkem is a game developed by 135 Tech Labs for the Android platform. It has AceSDK integrated with it. It is a time based game with each game session lasting 60 seconds. The game board has 36 dots of 6 different colors randomly arranged in 6 columns and 6 rows.

2.1 Overview

2.1.1 How to play?

The objective of the game is to connect as many adjacent dots of the same color as possible within 60 seconds. A dot can be linked to another of the same color as long as it is horizontally or vertically adjacent to it. A single link can consist of as many same colored adjacent dots as possible. When the user creates a link, the dots in the link disappear and their place is occupied by the dots above them. Each dot in a successful link contributes 1 point to the user score. When a closed loop is formed all the dots of similar color on the board disappear adding as many points to the user score as the same colored dots.

2.1.2 Integration points for AceSDK

At the end of every 60 seconds game session, the score is reported to AceSDK. Based on the score accumulated over the different sessions, Linkem notifies AceSDK to display the rewards to the user.

The steps used for the integration of the AceSDK with Linkem were:

1. Signing up as a developer on Hangoutt Rewards platform. This can be done at: `https://design.hangoutt.com/portal/signup.php`.
2. Importing AceSDK into workspace. This is explained in detail in Chapter 1.
3. Declaring AceSDK's AceActivity in Linkem's AndroidManifest.xml and passing the client credentials for Linkem. Refer Chapter 1 for details.
4. Connecting to AceSDK from the Launcher Activity.
5. Reporting the score to AceSDK after each session.
6. Rewarding the user.

2.2 Connecting to AceSDK

The Launcher Activity of Linkem is called StartActivity. It is in the OnCreate method of the StartActivity that the `connect(Activity activity)` method of AceSDK is invoked. The code snippet is shown in Figure 2.1.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.start_activity);

    //some code
    HangouttRewards.INSTANCE.connect(StartActivity.this);
    //some code
}
```

Figure 2.1: Connecting to AceSDK in the launcher activity.

2.3 Reporting the score

The Activity that handles the game display is called LinkemGame. This activity has a method called `gameOver(int score)` which is invoked when the 60

seconds are over, passing the current session score of the user as a parameter. Based on estimation that a use can score an average of 100-130 points per game session, the final score is converted to equivalent Hangoutt points. Following the benchmark of 100 Hangoutt points for a 10-minute interaction, the session score is divided by 10.

These equivalent Hangoutt points are reported to AceSDK as shown in Figure 2.2.

```
public void gameOver(int score) {
    //some code
    int hangoutt_points = score/10;

    HangouttRewards.INSTANCE.add(LinkemGame.this,
         String.valueOf(hangoutt_points));

    //some code
}
```

Figure 2.2: Reporting the score to AceSDK.

2.4 Rewarding the user

Once the score has been reported to AceSDK, the next activity called EndActivity is started. This activity invokes the `reward_user(Activity activity)` method of AceSDK. The code snippet for this invocation is shown in Figure 2.3.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    //some code
    HangouttRewards.INSTANCE.reward_user(EndActivity.this);
    //some code
}
```

Figure 2.3: Rewarding the user through AceSDK.

2.5 Improving the user experience

Linkem also allows the user to explicitly request for the rewards based on the previous sessions. This is handled by a fragment, called RewardFragment that is embedded in the StartActivity. It displays the user's accumulated score and provides a way to request for rewards based on this accumulated score.

This functionality allows the user to request for rewards based on the previous session scores without having to play a new game session. The fetchScore() method queries for the current accumulated score of the user by invoking the `get(Activity activity)` method of AceSDK. If the accumulated score is greater than a threshold score of 50 Hangoutt points, then a button called `get_rewards` is displayed on the screen. On clicking the `get_rewards` button, the `reward_user(Activity activity)` of AceSDK is invoked. The relevant code snippet is shown in Figure 2.4.

```
public class ScoreActivity extends Activity implements RewardsCountListener {

    Button get_rewards;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        //some code
        get_rewards = (Button)findViewById(R.id.rewards);
        get_rewards.setVisibility(View.INVISIBLE);

        HangouttRewards.INSTANCE.reward_user(ScoreActivity.this, true,
             this);
        //some code

        get_rewards.setOnClickListener(new OnClickListener() {

            @Override
                public void onClick(View v) {
                    HangouttRewards.INSTANCE.reward_user(ScoreActivity.this,
                        false, this);
                }
            });
        }

    @Override
    public void rewardsCount(int count) {
        if(count > 0) {
            get_rewards.setVisibility(View.VISIBLE);
        }
    }
    //some code
}
```

Figure 2.4: Fetching the user score through AceSDK.

2.6 Integration Effort Estimates

Table 2.1 lists the approximate time each step took while integrating AceSDK with Linkem.

Task	**Time required (approx.)**
Signing up	∼7 minutes
Importing AceSDK	∼4 minutes
Declaring AceActivity, the client credentials and the required permissions	∼6 minutes
Invoking AceSDK's methods	∼10 minutes
Testing and Error Handling	∼15 minutes

Table 2.1: Effort Estimates for AceSDK Integration.

Appendix A

Troubleshooting

1. If there is an error raised because of a difference in the versions of the library "android-support-v4.jar" in AceSDK and the application, please make sure that both the versions are same. Please copy the library from the application's Android Private Libraries into AceSDK's Android Private Libraries.
2. The `connect(Activity activity)` method has to be invoked by the application only on the `onCreate(Bundle savedState)` method of the launcher activity.
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`.