Unity Package Documentation

Version 1.0

135 Tech Labs

Contents

1 AceSDK Unity Package **4**

1.1 Overview 4

1.1.1 How to use AceSDK? 4

1.1.2 Where to use AceSDK? 5

1.2 Importing AceSDK Unity Package 5

1.2.1 Choosing an Interface 5

1.3 Configuration for a Scene 7

1.4 Hangoutt Singleton 9

1.5 Reporting the score 9

1.6 Querying the current score 9

1.7 Rewards for the user 10

1.8 Targeted Rewards for the user 10

1.9 Serendipitous Rewards 10

1.10 Hangoutt Points 12

1.11 Tracking effectiveness of reward moments 13

1.12 Querying for rewards before displaying 14

List of Figures

1.1 Importing the Unity Package. 6
1.2 Importing the uGUI interface package. 7
1.3 Importing the NGUI interface package. 8
1.4 Providing developer credentials. 8

List of Tables

1.1 Parameters that can be set for better reward targeting. 11
1.2 Significance of the parameters for the overloaded method. 12
1.3 Additional parameters for overrideParameters dictionary. 12

Chapter 1

AceSDK Unity Package

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

AceSDK Unity package implements the complete rewarding cycle for Hangoutt reward platform: 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 1.8.

1.1 Overview

1.1.1 How to use AceSDK?

1. Import AceSDK into the application (Section 1.2)
2. Configuration for a scene (Section 1.3)
3. Report the score of the user (Section 1.5)
4. Reward the user (Sections 1.7, 1.8 and 1.9)
5. (Optional) Get the user's score (Section 1.6)

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 Unity Package

The AceSDK Unity package has to be imported into the same project as the third-party application implementing the SDK. To do this, you have to select the downloaded AceSDK unity package as follows from the unity menu bar for your application project:

```
Assets --> Import Package --> Custom Package
```

This is illustrated in Figure 1.1.

The AceSDK unity package has support for NGUI and uGUI interface systems. Depending on which of these systems is used by your application, you will have to configure the corresponding GUI for AceSDK as explained in Section 1.2.1.

1.2.1 Choosing an Interface

uGUI is the default interface included with Unity 4.6 and 5+. NGUI is also a standard third-party interface plugin, widely used in unity projects. Upon a successful import of the AceSDK Unity package, both the NGUI and uGUI packages for AceSDK will be available for import in the Hangoutt folder. Please import only one of the interface packages depending on which one is supported by your application.

uGUI Interface

To extract the Hangoutt uGUI interface, double click on the "Interface (uGUI)" package located in the Hangoutt folder under Assets. In the subsequent menu

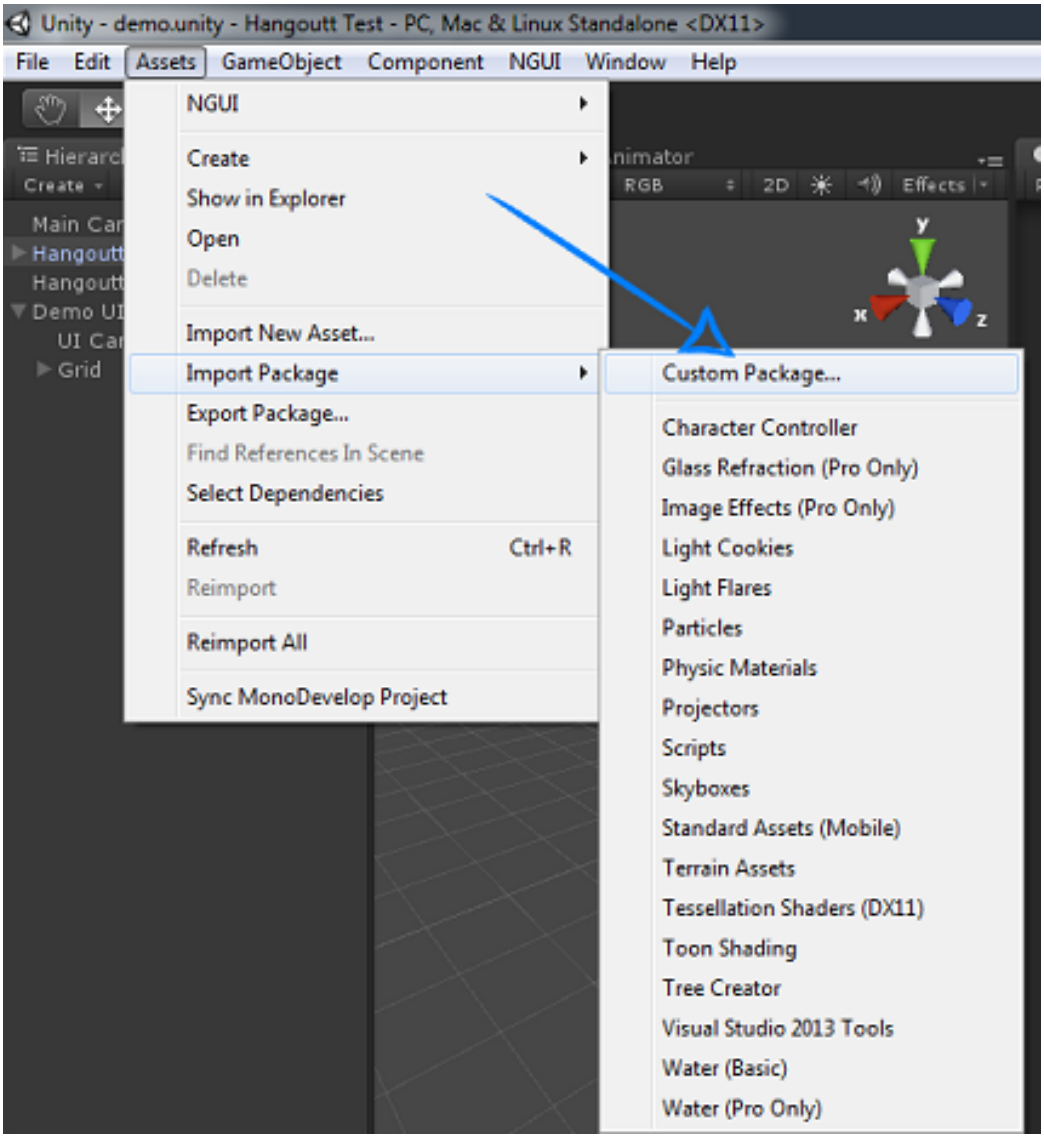

Figure 1.1: Importing the Unity Package.

which opens up, choose to import all files. This is illustrated in Figure 1.2.

(Note: The Hangoutt interface is expected to overlay your existing interface elements. Please ensure the "Sort Order" found on the Hangoutt uGUI canvas is higher than your existing canvas. The default sort order for Hangoutt is 99.)

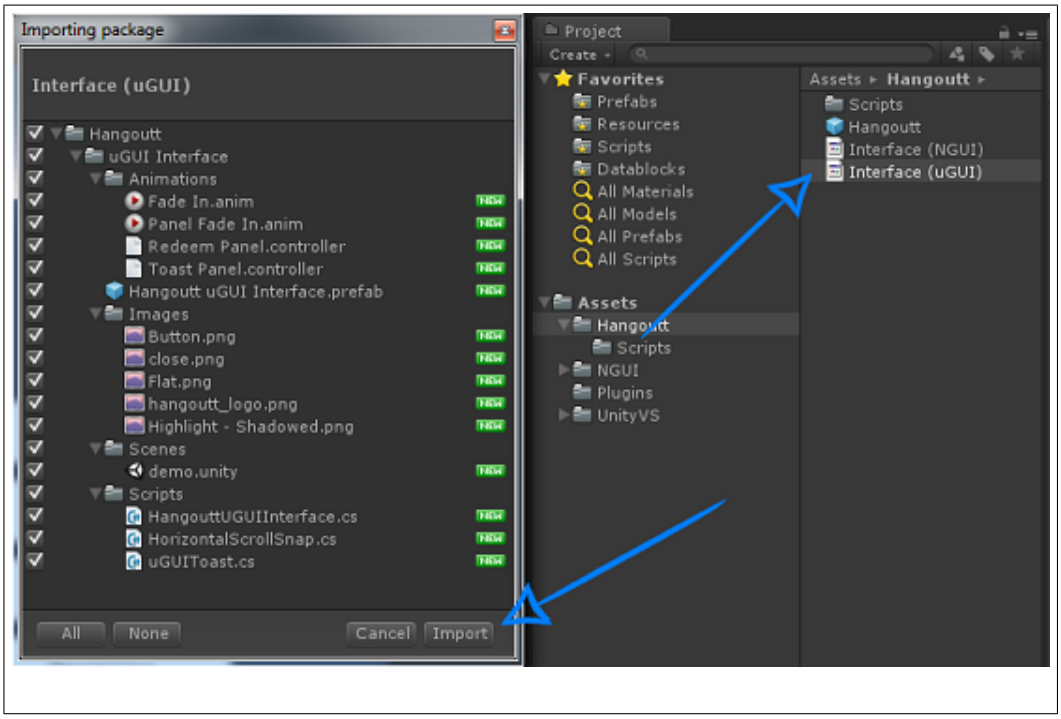

Figure 1.2: Importing the uGUI interface package.

NGUI Interface

Please ensure you have already installed NGUI before extracting this package. To extract the Hangoutt NGUI interface, double click on the "Interface (NGUI)" package located in the Hangoutt folder under Assets. In the subsequent menu which opens up, choose to import all files. This is illustrated in Figure 1.3.

1.3 Configuration for a Scene

In order to configure AceSDK for a scene, two prefabs are required to be added to the scene. The main "Hangoutt" prefab, and the interface prefab that corresponds to your chosen GUI system.

In addition to adding the prefabs, you also need to set the `client_id` and `client_secret`. To do this, select the Hangoutt object from your scene view. In the inspector, you will see fields to set your developer credentials. This is illustrated in Figure 1.4.

(Note: You will get your `client_id` and `client_secret` from the developer portal,

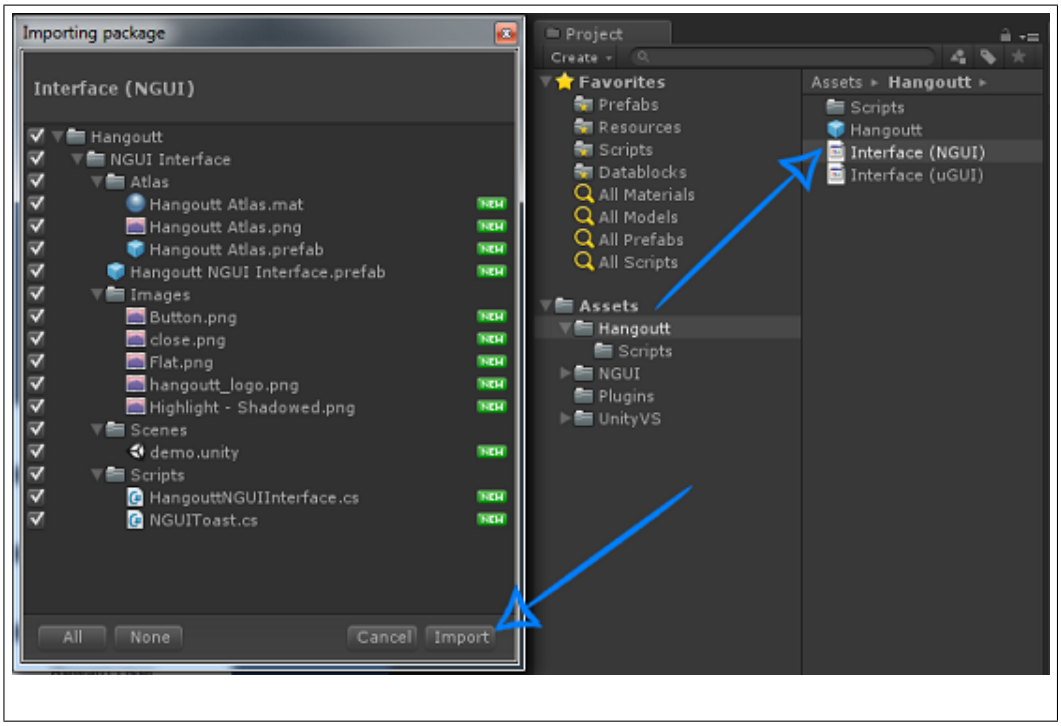

Figure 1.3: Importing the NGUI interface package.

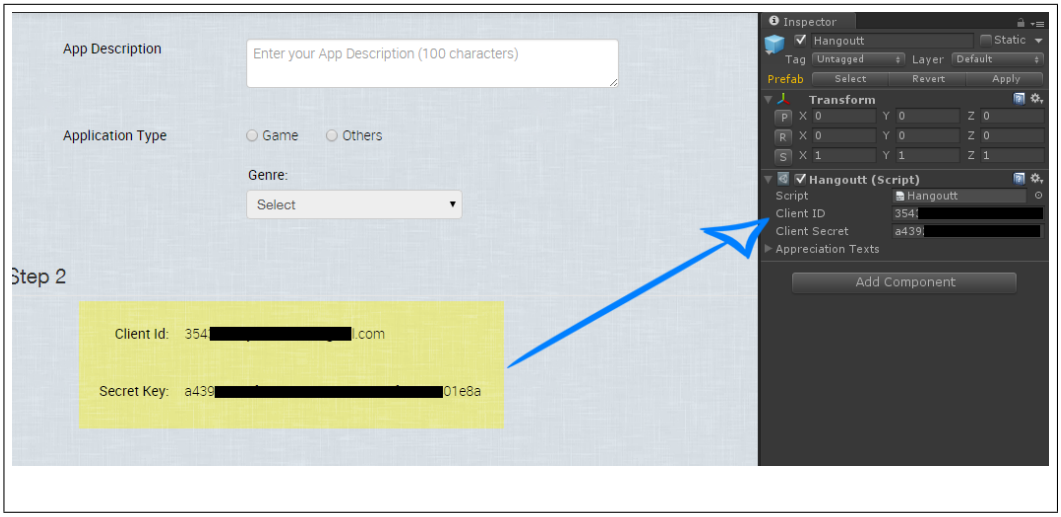

Figure 1.4: Providing developer credentials.

once you have added the application.)

1.4 Hangoutt Singleton

The AceSDK unity package provides a single-point interface of invoking methods through the Hangoutt singleton. In order to access the singleton, the namespace for AceSDK has to be imported in to the corresponding script. This can be done in the following manner:

```
using HangouttUnitySDK;
```

The Hangoutt singleton can be accessed with the following method:

```
Hangoutt.Main
```

1.5 Reporting the score

This method helps accumulate the score of the user over several invocations of the application. The parameter that needs to be passed to this method is the newly earned score that is to be reported. This method can be invoked in the following manner:

```
Hangoutt.Main.add("50");
```

The score that is reported is in terms of Hangoutt points earned. Please refer to Section 1.10 for a detailed explanation on how Hangoutt points are computed.

1.6 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.

```
score = Hangoutt.Main.get();
```

1.7 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 can be invoked as shown below.

```
Hangoutt.Main.reward_user();
```

Calling this method will fetch the available rewards based on the user's current score. If any are found, the rewards interface will be displayed to the user.

1.8 Targeted Rewards for the user

For applications that can specify a finer user profile for better reward targeting, the `SetUserField` method can be used to specify such properties. The set of parameters that can be set are listed in Table 1.1. The additional set of parameters are used to control the categorization and ranking of rewards on the server side.

The `SetUserField` method can be invoked as follows:

```
Hangoutt.Main.SetUserField(key, value);
```

This can be done anytime before fetching the rewards. The `SetUserField` method 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 1.11.

1.9 Serendipitous Rewards

There are instances where the user has made a great achievement but will not necessarily have enough points to earn a reward. In those instances you can display the rewards to user while ignoring their current score. Any redeemed rewards will also not deduct from their score.

Key(String)	Value(String)
`country_code`	2 letter ISO defined Country Code (in uppercase)
`country`	Country Name (lowercase)
`city`	City Name (lowercase)
`state`	State Name (lowercase)
`email`	User email id
`gender`	Gender("male", "female" or "all")
`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.

The following version of `reward_user` method has to be invoked for getting serendipitous rewards. An explanation of the parameters of this method is given in Table 1.2.

```
reward_user(bool count_only,
        bool serendipity,
        Dictionary<string, string> overrideParameters,
        Action<int> rewardsCountCallback);
```

In order to get serendipitous rewards, invoke the method as follows:

```
Hangoutt.Main.reward_user(false, true,
        overrideParameters);
```

For the overrideParameters dictionary, all parameters from Table 1.1 can be set. In addition, the parameters listed in Table 1.3 can be set:

Parameter	Significance
`count_only`	If set to true, retrieves the number of rewards currently available to the user. No interface will be displayed. The result will be passed to the rewardsCountCallback.
`serendipity`	If set to true will display rewards without factoring in the user's current score. Any redeemed offer will not subtract from the user's score. The score will have to be specified in overrideParameters.
`overrideParameters`	List of parameters to use when fetching rewards. Parameter values in this list will override any values previously set with SetUserField.
`rewardsCountCallback`	If set, and if `count_only` is true, will be called after fetching the rewards and contains a single integer parameter with the number of rewards available.

Table 1.2: Significance of the parameters for the overloaded method.

Key(String)	Value(String)
`limit`	Maximum number of rewards to be fetched, between 1 and 5 (default is 5).
`points`	Overrides the user's score when serendipity is true (default is 100).

Table 1.3: Additional parameters for overrideParameters dictionary.

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

1.11 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., Class 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. This can be set using either the `SetUserField` method or by using the `overrideParameters` dictionary.

The JSONArray node can be created as follows:

```
HangouttJSON.JSONNode labels =
        HangouttJSON.JSON.Parse ("[]");
labels[0] = "ClassName.FunctionName";
labels[1] = "Level2";
```

In order to specify labels through the `SetUserField` method, do the following:

```
Hangoutt.Main.SetUserField("labels", labels.ToString());
```

If instead, you would like to use the `overrideParameters` dictionary, do the following:

```
overrideParameters.Add("labels", labels.ToString());
```

1.12 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 rewardsCountCallback 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 rewardsCountCallback. 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 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.

```
Hangoutt.Main.reward_user(true,
        rewardsCount => {
                Debug.Log("Reward count is: "
                        + rewardsCount );
        }
);
```