

Android Native Plugin

[Short Overview](#)

[Billing](#)

[Billing Set Up](#)

[Classes Documentation](#)

[Play Service](#)

[Play Service Set Up](#)

[Classes Documentation](#)

[Google Cloud Set Up](#)

[Classes Documentation](#)

[Native Alerts](#)

[Use Example](#)

[Example Scenes](#)

[Support](#)

Shot Overview

This plugin, will provide easy and flexible functionality of Android native functions with are not available from clean Unity. (In-app purchases, native alerts, etc).

Billing Set Up

Make sure that [androidnative.jar](#) and [AndroidManifest.xml](#) is inside your **Assets/Plugins/Android** folder.

To implement in-apps in your application you should create new android application in google developer console and pass some info to the plugin. See instructions below how to set up and run billing example scene.

1) Create new Application in Google Developer Console and get **public license key**. See the step below:

1. Go to the [Google Play Developer Console](#) site and log in. You will need to register for a new developer account, if you have not registered previously. To sell in-app items, you also need to have a [Google Wallet](#) merchant account.
2. Click on **Try the new design** to access the preview version of the Developer Console, if you are not already logged on to that version.
3. In the **All Applications** tab, add a new application entry.
 1. Click **Add new application**.
 2. Enter a name for your new In-app Billing application.
 3. Click **Prepare Store Listing**.
4. In the **Services & APIs** tab, find and make a note of the public license key that Google Play generated for your application. This is a Base64 string that you will need to include in your application code later.

Your application should now appear in the list of applications in Developer Console.

2) Pass **public license key** to the plugin

- Open [PaymnetManagerExample](#) class, with is located under [Assets/Extensions/AndroidNative/Example/PaymnetManagerExample.cs](#).
- Assign your public key to the [base64EncodedPublicKey](#) variable.

Security Recommendation: It is highly recommended that you do not hard-code the exact public license key string value as provided by Google Play. Instead, you can construct the whole public license key string at runtime from substrings, or retrieve it from an encrypted store, before passing it to the plugin. This approach makes it more difficult for malicious third-parties to modify the public license key string in your APK file.

Setting Up for Test Purchases

To test your In-app Billing implementation with actual in-app purchases, you will need to register at least one test account on the Google Play Developer Console. You cannot use your developer account to test the complete in-app purchase process because Google Wallet does not let you buy items from yourself. If you have not set up test accounts before, see [Setting up test accounts](#).

Also, a test account can purchase an item in your product list only if the item is published. The application does not need to be published, but the item does need to be published.

To test your In-app Billing implementation with actual purchases, follow these steps:

1. **Upload your application as a draft application to the Developer Console.**
2. You do not need to publish your application to perform end-to-end testing with real product IDs; you only need to upload your application as a draft application. However, you must sign your application with your release key before you upload it as a draft application. Also, the version number of the uploaded application must match the version number of the application you load to your device for testing. To learn how to upload an application to Google Play, see [Uploading applications](#).
3. **Add items to the application's product list.**
4. Make sure that you publish the items (the application can remain unpublished). See [Creating a product list](#) to learn how to do this.
5. **Install your application on an Android-powered device.**
6. You cannot use the emulator to test In-app Billing; you must install your application on

a device to test In-app Billing.

7. **Verify that your device is running a supported version of the Google Play application or the MyApps application.**
8. If your device is running Android 3.0, In-app Billing requires version 5.0.12 (or higher) of the MyApps application. If your device is running any other version of Android, In-app Billing requires version 2.3.4 (or higher) of the Google Play application. To learn how to check the version of the Google Play application, see [Updating Google Play](#).
9. **Make in-app purchases in your application.**

Note: The only way to change the primary account on a device is to do a factory reset, making sure you log on with your primary account first.

When you have finished testing your In-app Billing implementation, you are ready to publish your application on Google Play. You can follow the normal steps for [preparing](#), [signing](#), and [publishing on Google Play](#).

Signing and Uploading apk with Unity

To be able to create in-app purchases you should upload your apk file to the developer console. Apk must be signed with your private key. By default when you build apk file with Unity, it signed with the debug key. It means that it not suitable for upload to the google developer console.

There is a lot of ways how you can create private key for your application, you can read more [here](#) about android application signing, or use [Unity build in tools](#):

Next step is app configuration.

You have to choose your bundle bundle ID

A bundle ID otherwise known as a **package** in Android is the unique identifier for all Android apps. It needs to be unique as when you upload it to Google Play it identifies and publishes your app use the package name as the unique app identification.

Really it is the only thing which is necessary to identify your app, and generally it has 3

parts:

com.example.testapp

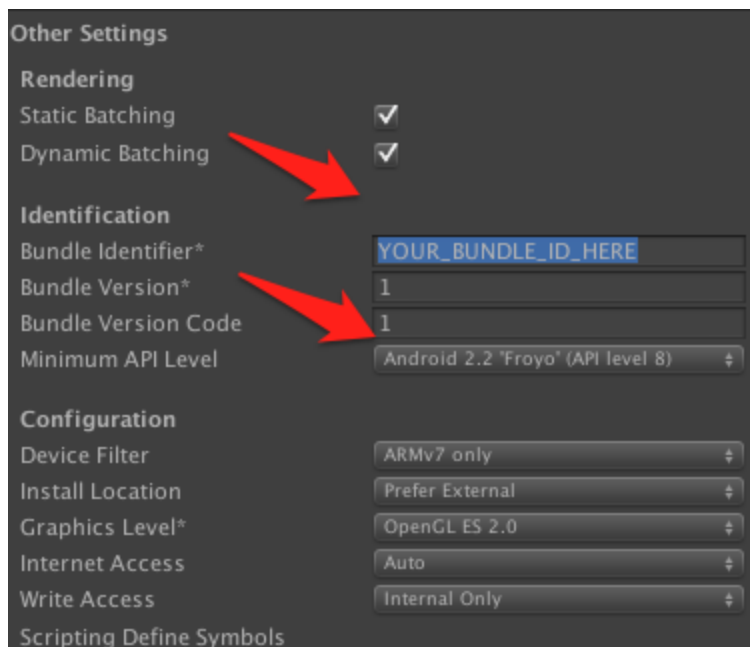
Where **example** is generally the company/publishers name, and **testapp** is the app name.

You will not be able to upload an APK to the store which has the same package as another app already in the store.

When your bundle ID is ready add it to the Unity application setting and to the [AndroidManifest.xml](#).

Also Plugin use version 3 of Android In-app billing. This is the latest Android billing API and it requires minimum [Android 2.2.x\(FROYO\)](#) **SDK int 8** or higher.

Here is screenshot of required setting in Unity.



And the [AndroidManifest.xml](#) settings

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     android:installLocation="preferExternal"
4     package="YOUR_BUNDLE_ID_HERE"
5     android:versionName="2.0"
6     android:versionCode="2">
7
8     <supports-screens android:smallScreens="true" android:normalScreens="true" android:large
9
10    <application
11        android:icon="@drawable/app_icon"
12        android:label="@string/app_name"
13        android:debuggable="false">
14
15        <activity android:name="com.android.MainActivity" android:label="@string/app_name" and
16            <intent-filter>
17                <action android:name="android.intent.action.MAIN" />
18                <category android:name="android.intent.category.LAUNCHER" />
19            </intent-filter>
20        </activity>
21        <activity android:name="com.unity3d.player.VideoPlayer" android:label="@string/app_nam
22        </activity>
23
24    </application>
25    <uses-feature android:glEsVersion="0x00020000" />
26    <uses-sdk
27        android:minSdkVersion="8"
28        android:targetSdkVersion="15" />
29    <uses-permission android:name="android.permission.INTERNET" />
30    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
31
32    <!-- VERY IMPORTANT! Don't forget this permission, or in-app billing won't work. -->
33    <uses-permission android:name="com.android.vending.BILLING" />
34
35 </manifest>

```

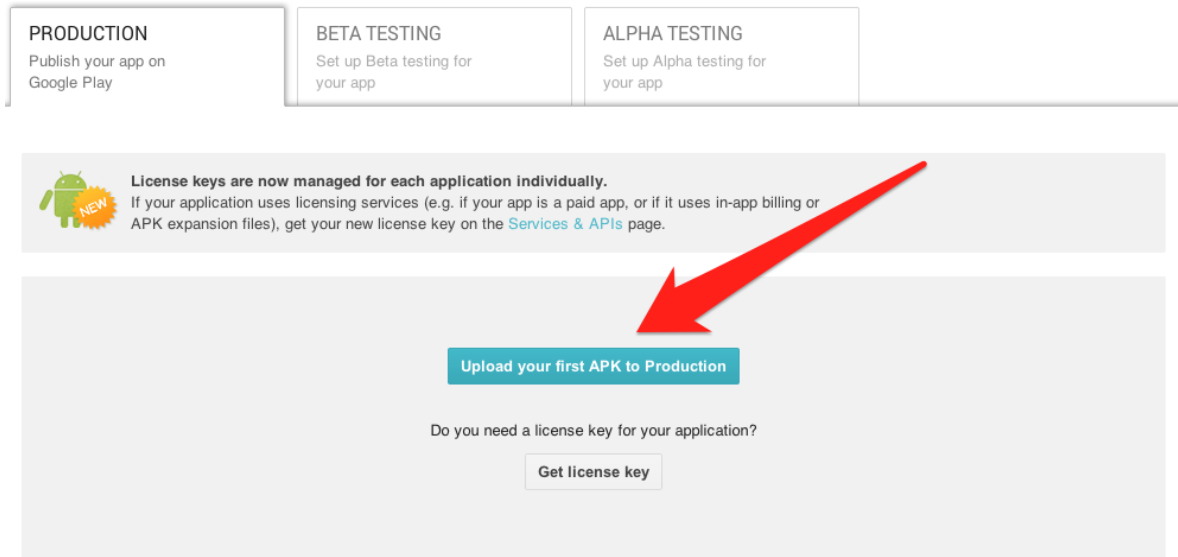


You can build your signed apk file now. Just press **build** button.

Note: You should have latest android SDK on your computer, to make Unity able build apk file.

Note: Android plugin should be included to your application, if you will build signed application without plugin included, application will not have permissions to use billing.

After signed apk is created you can upload it to the Google. Choose your created application on Google Developer Console, open APK tab and press “**Upload your First APK to Production**” button.



After apk is uploaded you can start testing exemple scene and try to modify [PaymnetManagerExample](#) class to work with your products, or create your own using [PaymnetManagerExample](#) as example.

Classes Documentation

Billing module

[InAppPurchaseManager](#) class.

API methods:

Add product's SKU, with will be registered after billing initialization, do this before calling loadStore function

```
public void addProduct(string SKU)
```

Connecting to the Android Market with your public key

```
public void loadStore(string base64EncodedPublicKey)
```

Get's registred products details

```
public void retrieveProducDetails()
```

Purchase the product

```
public void purchase(string SKU)
```

Consume the product

```
public void consume(string SKU)
```

Getters:

Current inventory the product

```
public AndroidInventory inventory
```


Events:

Fires when purchase product flow end's with success or fail. Event data contains [BillingResult](#).

[ON_PRODUCT_PURCHASED](#)

Fires when consume product flow end's with success or fail. Event data contains [BillingResult](#).

[ON_PRODUCT_CONSUMED](#)

Fires when billing is connected. Event data contains [BillingResult](#).

[ON_BILLING_SETUP_FINISHED](#)

Fires when product details are loaded. Event data contains [BillingResult](#).

[ON_RETRIEVE_PRODUC_FINISHED](#)

[AndroidInventory class.](#)

Methods:

Returns true if current user owns the product and false if not

[public bool](#) [IsProductPurchased](#)([string](#) SKU)

Get's product details by SKU

[public](#) [ProductTemplate](#) [GetProductDetails](#)([string](#) SKU)

Get's purchase details by SKU

[public](#) [PurchaseTemplate](#) [GetPurchaseDetails](#)([string](#) SKU)

List of customer purchases

```
public List<PurchaseTemplate> purchases
```

List of registered products

```
public List<ProductTemplate> products
```

ProductTemplate class

Getters:

item id

```
public string SKU
```

Localized item price string

```
public string price
```

Localized item title string

```
public string title
```

Localized item description string

```
public string description
```

PurchaseTemplate class

Getters:

purchase order id

```
public string orderId
```

purchase package name

```
public string packageName
```

purchased item id

public string SKU

developer payload of purchase

public string developerPayload

BillingResult class

Contains information about purchase. NULL when fires with events like ON_BILLING_SETUP_FINISHED or ON_RETRIEVE_PRODUC_FINISHED, or when result is failed.

public PurchaseTemplate purchase

contains response code. See the [BillingResponseCodes](#) class for more info.

public int response

contains response message

public string message

true response was succeed.

public bool isSuccess

true response was failed

public bool isFailure

BillingResponseCodes class

public const int BILLING_RESPONSE_RESULT_OK = 0;

public const int BILLING_RESPONSE_RESULT_USER_CANCELED = 1;

public const int BILLING_RESPONSE_RESULT_BILLING_UNAVAILABLE = 3;

public const int BILLING_RESPONSE_RESULT_ITEM_UNAVAILABLE = 4;

```
public const int BILLING_RESPONSE_RESULT_DEVELOPER_ERROR = 5;
public const int BILLING_RESPONSE_RESULT_ERROR = 6;
public const int BILLING_RESPONSE_RESULT_ITEM_ALREADY_OWNED = 7;
public const int BILLING_RESPONSE_RESULT_ITEM_NOT_OWNED = 8;

// Helper error codes
public const int BILLINGHELPER_ERROR_BASE = -1000;
public const int BILLINGHELPER_REMOTE_EXCEPTION = -1001;
public const int BILLINGHELPER_BAD_RESPONSE = -1002;
public const int BILLINGHELPER_VERIFICATION_FAILED = -1003;
public const int BILLINGHELPER_SEND_INTENT_FAILED = -1004;
public const int BILLINGHELPER_USER_CANCELLED = -1005;
public const int BILLINGHELPER_UNKNOWN_PURCHASE_RESPONSE = -1006;
public const int BILLINGHELPER_MISSING_TOKEN = -1007;
public const int BILLINGHELPER_UNKNOWN_ERROR = -1008;
public const int BILLINGHELPER_SUBSCRIPTIONS_NOT_AVAILABLE = -1009;
public const int BILLINGHELPER_INVALID_CONSUMPTION = -1010;
```

Play Service Set Up

Before you begin

- You should have your Android development environment set up.
- You should have a physical device running Android 2.2 (Froyo) or higher for testing.

Step 1: Check / Download all necessary files and prepare your device

Make sure that you have all listed files at this location

Assets/Plugins/Android/AndroidManifest.xml

Assets/Plugins/Android/AndroidNative.jar

Assets/Plugins/Android/libs/android-support-v4.jar

Assets/Plugins/Android/libs/google-play-services.jar

Assets/Plugins/Android/res/values/ids.xml

To install the Google Play services SDK for development:

1. Launch the SDK Manager.
 - On Windows, double-click the [SDK Manager.exe](#) file at the root of the Android SDK directory.
 - On Mac or Linux, open a terminal and navigate to the [tools/](#) directory in the Android SDK, then execute [android sdk](#).
2. Install the Google Play services SDK.
3. Scroll to the bottom of the package list, expand Extras, select Google Play services, and install it.
4. The Google Play services SDK is saved in your Android SDK environment at [<android-sdk>/extras/google/google_play_services/](#).
5. Install a compatible version of the Google APIs platform.
6. If you want to test your app on the emulator, expand the directory for Android 4.2.2 (API 17) or a higher version, select Google APIs, and install it. Then create a new [AVD](#) with Google APIs as the platform target.
7. **Note:** Only Android 4.2.2 and higher versions of the Google APIs platform include Google Play services.

Step 2: Set up the game in the Developer Console

The Google Play Developer Console is where you manage game services for your game, and configure metadata for authorizing and authenticating your game.

To set up the sample game in the Developer Console:

1. Point your web browser to the [Developer Console](#), and sign in. If you haven't

registered for the Developer Console before, you will be prompted to do so.

2. Follow these instructions to [add your game to the Developer Console](#).
 - a. When asked if you use Google APIs in your app, select **I don't use any Google APIs in my game yet**.
 - b. For the purpose of this training, you can fill up the form with your own game details. For convenience, you can use the placeholder icons and screenshots provided in the [Downloads](#) page.
3. Follow these instructions to [generate an OAuth 2.0 client ID](#) for your Android app.
 - a. When linking your Android app, make sure to specify the exact package name you used previously when renaming sample package.
 - b. You can use the Unity to generate a new keystore and signed certificate if you don't have one already. To learn how to generate a new keystore and signed certificate, see [Compile and sign with Unity](#).
4. Make sure to record the following information for later:
 - a. Your [application ID](#): This is a string consisting only of digits (typically 12 or more), at the beginning of your client ID.
 - b. Your signing certificate: Note which certificate you used when setting up your API access (the certificate whose SHA1 fingerprint you provided). You should use the same certificate to sign your app when testing or releasing your app.
5. Configure achievements for Test Scene Challenge:
 - a. Select the **Achievements** tab in the Developer Console.
 - b. Add the following sample achievements:

Name	Description	Special Instructions
Prime	Get a score that's a prime number.	None
Humble	Request a score of 0.	Make this a hidden achievement.
Bored	Play the game 10 times.	Make this an incremental achievement with 10 steps to unlock.

- c. Record the IDs (long alphanumeric strings) for each achievement that you created.

- d. Configure achievements that are appropriate for your game. To learn more, see the [concepts behind achievements](#).
6. Configure the leaderboards for Test Scene:
 - a. Select the the **Leaderboards** tab in the Developer Console.
 - b. Add two sample leaderboards: one named “Easy High Scores” and another named “Hard High Scores”. Both leaderboards should use Integer score formatting with 0 decimal places, and an ordering type of **Larger is better**.
 - c. Record the IDs (long alphanumeric strings) for each leaderboard you created.
 - d. Configure leaderboards that are appropriate for your game. To learn more, see the [concepts behind leaderboards](#).
7. [Add test accounts for your game](#). This step is needed only for apps that have not yet been published in the Developer Console. Before the app is published, only the test accounts listed in the Developer Console can log in. However, once an application is published, everyone is allowed to log in.

Step 3: Modify your code

To run the game, you need to configure the application ID as a resource in your Android project. You will also need to add games metadata in the [AndroidManifest.xml](#).

1. Open **Assets/Plugins/Android/res/values/ids.xml** and replace the placeholder IDs.
 - a. Specify your application ID in the [app_id](#) resource.
 - b. Specify each achievement ID that you created earlier in the corresponding [achievement_*](#) resource.
 - c. Specify each leaderboard ID that you created earlier in the corresponding [leaderboard_*](#) resource.
2. Open [AndroidManifest.xml](#) and enter your package name in the [package](#) attribute of the `<manifest>` element.

Step 4: Test your game

To ensure that game services are functioning correctly in your game, test the application

before you publish it on Google Play.

Note: It's recommended that you test on a physical Android device. However, if you do not have a physical device, you can test against the [Android Emulator](#). To do so, download the emulator system image that includes the Google Play Services, under **Android 4.2.2**, from the [SDK Manager](#).

To run your game on your physical test device:

1. Verify that you have set up the test account that you are using to log in to the app (as described in [Step 2](#)).
2. Export an APK and sign it with the same certificate that you used to set up the project in Developer Console.
3. Install the signed APK on your physical test device.

Classes Documentation

Play Service

[GooglePlayConnection](#) : Singleton<GooglePlayConnection> class.

API methods:

should be called on application start. It will create connection to the play service and sign in user if user was signed before. Best practice to call it only once. Any way other calls will be ignored by the plugin.

*if you want to use only Game service (Leader-boards, achievements) use
`GooglePlayConnection.CLIENT_GAMES`*

*if you want to use only Google Cloud service use
`GooglePlayConnection.CLIENT_APPSTATE`*

*if you want both: `GooglePlayConnection.CLIENT_GAMES` |
`GooglePlayConnection.CLIENT_APPSTATE`*

and if you want to get available permissions for your app use:

GooglePlayConnection.CLIENT_ALL

public void start(**int** clientsToUse)

Connect to Play Service

public void connect()

Disconnect from Play Service

public void disconnect()

Getters:

Current connection state

public static GPConnectionState state

Events:

Fires when GooglePlayManager state is CONNECTED. Event data null;

PLAYER_CONNECTED

Fires when GooglePlayManager state is DISCONNECTED. Event data null;

PLAYER_DISCONNECTED

GooglePlayManager : Singleton<GooglePlayManager> class.

API methods:

Should be called on application start. It will create connection to the play service and will sign in user if user was signed before.

`public void start()`

Connect to Play Service

`public void connect()`

Disconnect from Play Service

`public void disconnect()`

Show default Google Play Achievements UI

`public void showAchivmentsUI()`

Show default Google Play Leaderboards UI

`public void showLeaderBoardsUI()`

Show Leader board by name or id

`public void showLeaderBoard(string leaderboardName)`

`public void showLeaderBoardsUI(string leaderboardId)`

Trigger player info request, PLAYER_LOADED event will be fired on complete

`public void loadPlayer()`

Trigger submit score request, SCORE_SUBMITTED event will be fired on complete

`public void submitScore(string leaderboardName, int score)`

`public void submitScoreByld(string leaderboardId, int score)`

Trigger leaderboards info request, LEADERBOARDS_LOEADED event will be fired on complete

`public void loadLeaderBoards()`

Trigger achievement report request, ACHIEVEMENT_UPDATED event will be fired on complete

`public void reportAchievement(string achievementName)`

```
public void reportAchievementByld(string achievementId)
```

Trigger achievement reveal request, ACHIEVEMENT_UPDATED event will be fired on complete

```
public void revealAchievement(string achievementName)
```

```
public void revealAchievementByld(string achievementId)
```

Trigger achievement increment request, ACHIEVEMENT_UPDATED event will be fired on complete

```
public void incrementAchievement(string achievementName, int numsteps)
```

```
public void incrementAchievementByld(string achievementId, int numsteps)
```

Trigger achievement info load request, ACHIEVEMENT_UPDATED event will be fired on complete

```
public void loadAchivments()
```

Public methods:

Get's Leader board by id

```
public GPLeaderBoard GetLeaderBoard(string leaderboardId)
```

Get's Achievement board by id

```
public GPAchievement GetAchievement(string achievementId)
```

Getters:

Information about current player

```
public GooglePlayerTemplate player
```

Loaded Leaderboards

```
public Dictionary<string, GPLeaderBoard> leaderBoards
```

Loaded Achievements

`public Dictionary<string, GPAchievement> achievements`

Events:

Fires on Leaderboard score submitted. Event data contains [GooglePlayResult](#).

`SCORE_SUBMITTED`

Fires on Leaderboards data loaded. Event data contains [GooglePlayResult](#).

`LEADERBOARDS_LOEADED`

Fires on current player data loaded. Event data contains [GooglePlayResult](#).

`PLAYER_LOADED`

Fires on when achievement was updated. Event data contains [GooglePlayResult](#).

`ACHIEVEMENT_UPDATED`

Fires on Achievements data loaded. Event data contains [GooglePlayResult](#).

`ACHIEVEMENTS_LOADED`

[GooglePlayResult](#) class.

Getters:

contains response result code

`public GooglePlayResponseCode response`

contains response message

`public string message`

true when result succeeded

`public bool isSuccess`

true when result is failed

`public bool isFailure`

Contain Leaderboards id

`public string leaderboardId`

Contain Achievement id

`public string achievementId`

GPAchievement class

Getters:

achievement id

`public string id`

achievement name

`public string name`

achievement description

`public string description`

achievement current steps, -1 for non-incremental achievement

`public int currentSteps`

achievement total steps, -1 for non-incremental achievement

`public int totalSteps`

achievement type

```
public GPAchievementType type
```

achievement state

```
public GPAchievementState state
```

GPLeaderBoard class

Methods:

Get's score

```
public int GetScore (GPCollectionType collection, GPBoardTimeSpan timeSpan)
```

Get's rank

```
public int GetRank (GPCollectionType collection, GPBoardTimeSpan timeSpan)
```

Get's score variant class

```
public LeaderBoardScoreVariant GetVariant (GPCollectionType collection,  
GPBoardTimeSpan timeSpan)
```

Getters:

Leaderboard id

```
public string id
```

Leader board title

```
public string name
```

List of Leaderboards player scores

```
public List<LeaderBoardScoreVariant> scores
```

GooglePlayerTemplate class

Getters:

player id

public string id

player name

public string name

LeaderBoardScoreVariant class

Getters:

rank

public int rank

score

public int score

collection type

public GPCollectionType collection

score time span

public GPBoardTimeSpan timeSpan

GPConnectionState class

```
enum {  
    STATE_UNCONFIGURED,  
    STATE_DISCONNECTED,  
    STATE_CONNECTING,  
    STATE_CONNECTED  
}
```

GPCollectionType class

```
enum {  
    COLLECTION_PUBLIC,  
    COLLECTION_SOCIAL  
}
```

GPBoardTimeSpan classes

```
enum {  
    TIME_SPAN_DAILY,  
    TIME_SPAN_WEEKLY,  
    TIME_SPAN_ALL_TIME  
}
```

GPAchievementType classes

```
enum {  
    TYPE_STANDARD,  
    TYPE_INCREMENTAL  
}
```

GPAchievementState classes

```
enum {  
    STATE_UNLOCKED,  
    STATE_REVEALED,  
    STATE_HIDDEN  
}
```

GooglePlayResponseCode classes

```
enum {  
    STATUS_OK,  
    STATUS_INTERNAL_ERROR,  
    STATUS_NETWORK_ERROR_OPERATION_DEFERRED,  
    STATUS_NETWORK_ERROR_NO_DATA,  
    STATUS_CLIENT_RECONNECT_REQUIRED,  
    STATUS_LICENSE_CHECK_FAILED,  
    STATUS_NETWORK_ERROR_STALE_DATA,  
    UNKNOWN_ERROR,  
  
    STATUS_ACHIEVEMENT_UNLOCKED,  
    STATUS_ACHIEVEMENT_UNKNOWN,  
    STATUS_ACHIEVEMENT_NOT_INCREMENTAL,  
    STATUS_ACHIEVEMENT_UNLOCK_FAILURE,  
  
    STATUS_STATE_KEY_NOT_FOUND,  
    STATUS_STATE_KEY_LIMIT_EXCEEDED  
}
```

Google Cloud Set Up

If you haven't already done so, please review the [Cloud Save](#) guide to familiarize yourself with the concepts behind saving a user's application state using this service.

Caution: Calls to [UpdateState\(\)](#) that result in a conflict do not immediately trigger a callback to [OnStateConflict\(\)](#). The Cloud Save service signals a conflict the next time your application requests [LoadState\(\)](#) by calling [OnStateConflict\(\)](#).

To enable use of the Cloud Save service in your application, make sure that [AndroidManifest.xml](#) contains following meta-data tag:

```
<manifest ...>
  <application ...>
    <meta-data android:name="com.google.android.gms.appstate.APP_ID"
      android:value="@string/app_id" />
    ...
  </application>
</manifest>
```

it should be there if you did not change the file after downloading the plug-in.
And of course you should do the same set up action as for [Play Service Set Up](#).

Classes Documentation

[GoogleCloudManager](#) : Singleton<[GoogleCloudManager](#)> class.

API methods:

Will load all saved states. [ALL_STATES_LOADED](#) event is triggered

[public void](#) loadAllStates()

This method updates the local copy of the app state and syncs the changes to the server. If the local data conflicts with the data on the server, this will be indicated the next time you call `LoadState`. `STATE_UPDATED` or `STATE_CONFLICT` event is triggered

```
public void updateState(int stateKey, string data)
```

Resolve a previously detected conflict in app state data. Note that it is still possible to receive a conflict callback after this call. This will occur if data on the server continues to change. In this case, resolution should be retried until a successful status is returned. `STATE_RESOLVED` or `STATE_CONFLICT` events is triggered

```
public void resolveState(int stateKey, string resolvedData, string resolvedVersion)
```

Delete the state data for the current app. This method will delete all data associated with the provided key, as well as removing the key itself. Note that this API is not version safe. This means that it is possible to accidentally delete a user's data using this API. For a version safe alternative, consider using `updateState` with empty data instead. `STATE_DELETED` event is triggered

```
public void deleteState(int stateKey)
```

Asynchronously loads saved state for the current app. `STATE_LOADED` event is triggered

```
public void loadState(int stateKey)
```

Get state data by key. Note that state should be loaded before you can access it data via this function.

```
public void GetStateData(int stateKey)
```

Getters:

Gets the maximum app state size per state key in bytes. Guaranteed to be at least 128 KB. May increase in the future.

`public int maxStateSize`

Gets the maximum number of keys that an app can store data in simultaneously.

`public int maxNumKeys`

Gets states dictionary

`public Dictionary<int, string> states`

Events:

Fires on state delete. Event data contains [GoogleCloudResult](#).

`STATE_DELETED`

Fires on state update. Event data contains [GoogleCloudResult](#).

`STATE_UPDATED`

Fires on state data loaded. Event data contains [GoogleCloudResult](#).

`STATE_LOADED`

Fires on state data resolved. Event data contains [GoogleCloudResult](#).

`STATE_RESOLVED`

Fires on state data conflict detected. Event data contains [GoogleCloudResult](#).

`STATE_CONFLICT`

Fires on all states data loaded. Event data contains [GoogleCloudResult](#).

`ALL_STATES_LOADED`

Warning: Do not use any function of this class before you connected to the play service.

GoogleCloudResult class.

Getters:

contains response result code

`public GooglePlayResponseCode response`

contains response message

`public string message`

true when result succeeded

`public bool isSuccess`

true when result is failed

`public bool isFailure`

state key

`public string stateKey`

Local state data

`public string stateData;`

conflicted data on server

`public string serverConflictData;`

resolved version

```
public String resolvedVersion;
```

Compile and sign with Unity

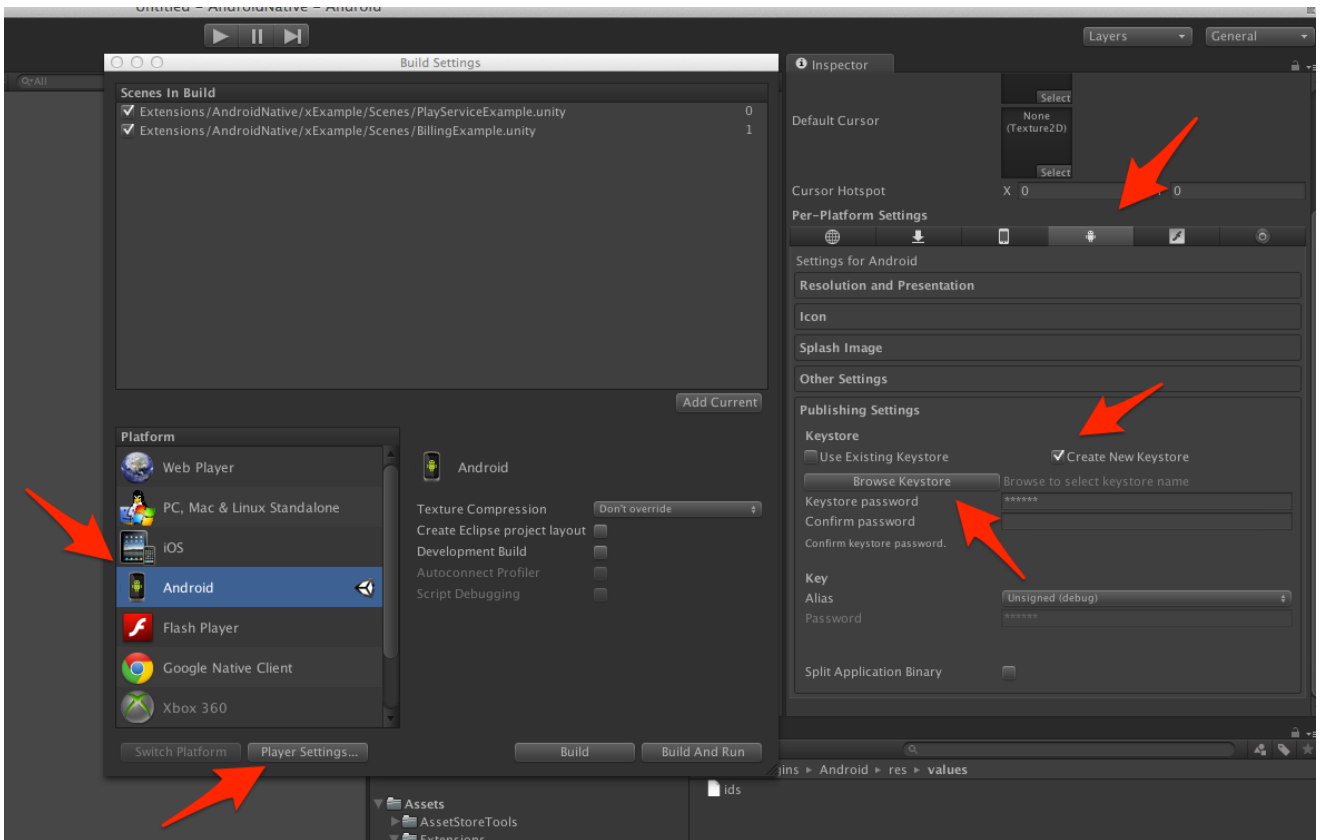
Go to:

File → Build Settings

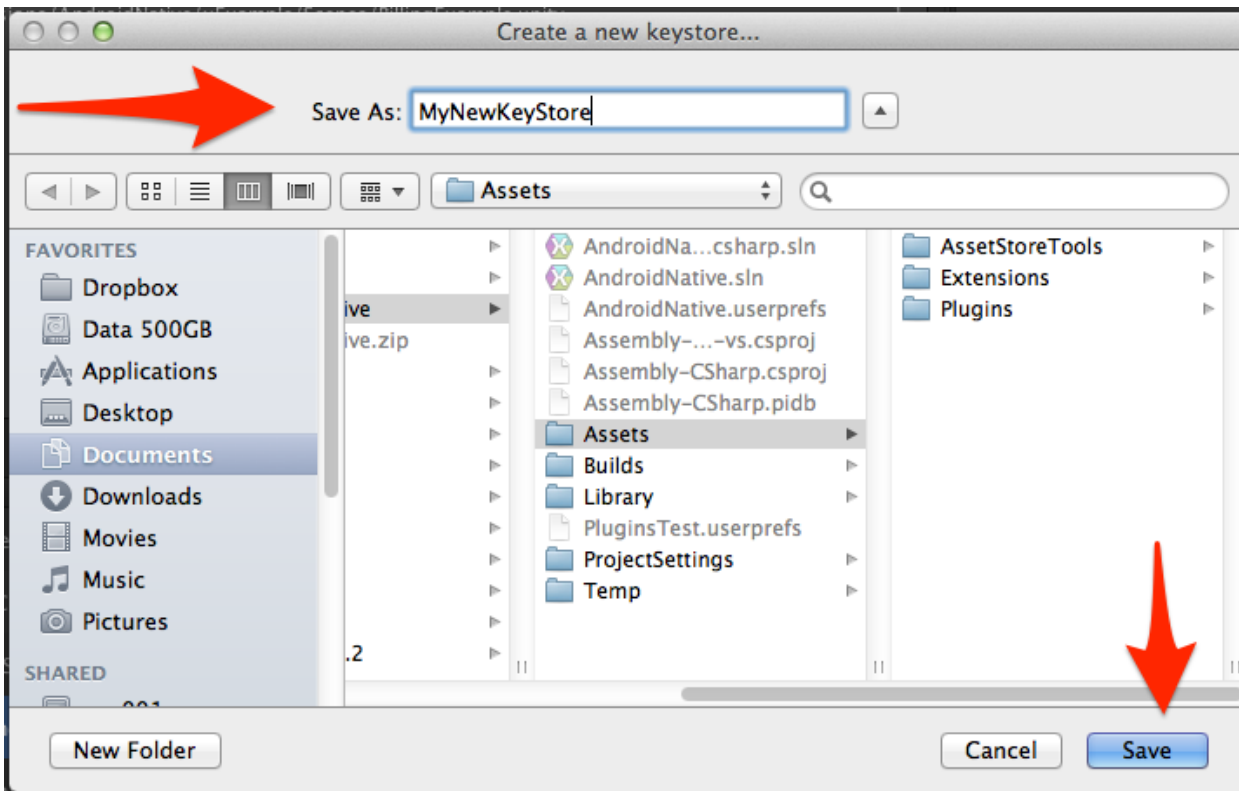
Choose Android platform and press **Player Settings** button.

In player settings navigate to the android tab, and choose **Other Setting** menu.

To generate new key store check “**Create New Keystore**” toggle and press “**Browse**” button.



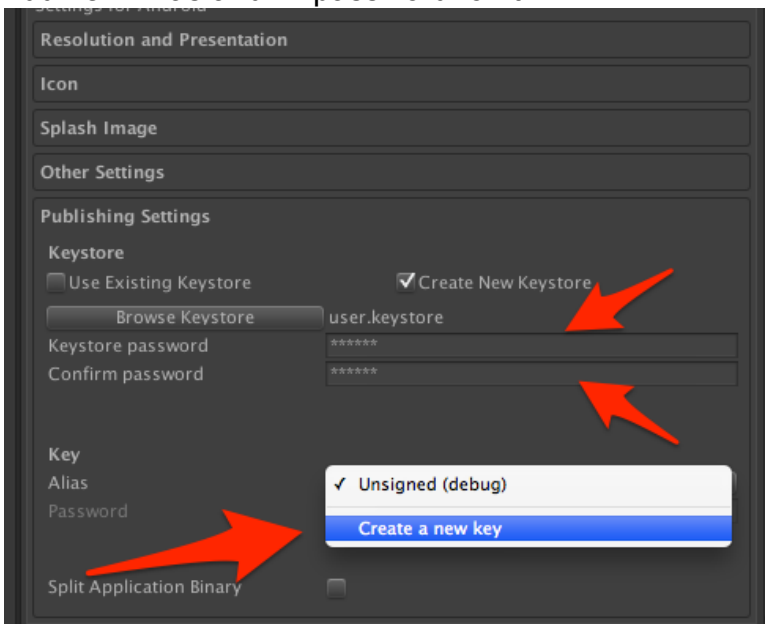
in the dialog box, select the path and name for the new keystore.



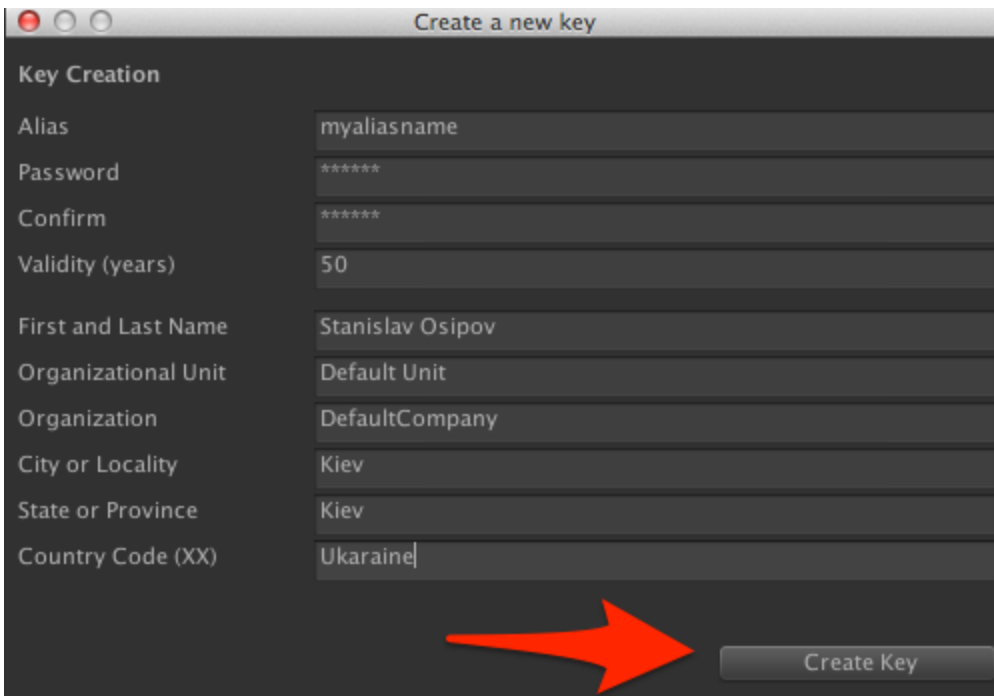
And press “**Save**” button.


Do not worry if keystore file not yet created. Fill the **Keystore password** and **Confirm Password** fields.

Add new Alias and fill password for it



Fill all data in the dialog window and press “**Create Key**” button.



Key Creation	
Alias	myaliasname
Password	*****
Confirm	*****
Validity (years)	50
First and Last Name	Stanislav Osipov
Organizational Unit	Default Unit
Organization	DefaultCompany
City or Locality	Kiev
State or Province	Kiev
Country Code (XX)	Ukraine
<div> <button>Create Key</button></div>	

Keystore and Alias for signing your app is created.

Make sure to record the package name and signing certificate that you configured in this step. Using a different certificate or package name in your application will cause authentication failures.

Warning: Keep your private key secure. Before you run Keytool, make sure to read [Securing Your Private Key](#) for a discussion of how to keep your key secure and why doing so is critically important to you and to users. In particular, when you are generating your key, you should select strong passwords for both the keystore and key.

Warning: Keep the keystore file you generate with Keytool in a safe, secure place. You must use the same key to sign future versions of your application. If you republish your app with a new key, Google Play will consider it a new app. For more information on settings that must remain constant over the life of your app, see the Android Developer Blog post [Things That Cannot Change](#).

[Learn more about app signing.](#)

Open a terminal, run the the Keytool utility to get the SHA-1 fingerprint of the certificate.

```
keytool -exportcert -alias <alias-name> -keystore <path-to-keystore> -list -v
```

You will need this SHA-1 fingerprint to [Generate an OAuth 2.0 client ID](#)

You can build signed application now. Simply go to:

File → **Build Settings**, choose Android platform and press build button. Then upload and install produced apk on your device.

Or if you have your device connected to the computer with “USB Debugging” option. You can use **File** → **Build and Run**.

Native Alerts

description of [AndroidRateUsPopUp](#), [AndroidMessage](#), [AndroidDialog](#)

Android Rate Pop Up

Pop up creation:

```
ndroidRateUsPopUp rate = AndroidRateUsPopUp.Create("Rate Us", rateText,
rateUrl);
```

Rate pop up will appear after this lines, if you want to listen rate pop up events you should add [COMPLETE](#) listener on it.

```
rate.addEventListener(BaseEvent.COMPLETE, OnRatePopUpClose);
```

example of [OnRatePopUpClose](#) function:

```
private void OnRatePopUpClose(CEvent e) {
    (e.dispatcher as
AndroidRateUsPopUp).removeEventListener(BaseEvent.COMPLETE, OnRatePopUpClose);
    string result = e.data.ToString();
    AndroidNative.showMessage("Result", result + " button pressed");
}
```

[AndroidDialogResult](#) result can contain: [RATED, REMIND, DECLINED](#) of [AndroidDialogResult](#) class.

Android Dialog Pop Up

Creation:

```
AndroidDialog dialog = AndroidDialog.Create("Dialog Titile", "Dialog
message");
```

Listeners:

```
dialog.addEventListener(BaseEvent.COMPLETE, OnDialogClose);
```

[onDialogClose](#) function example:

```
private void OnDialogClose(CEvent e) {
```

```

        //removing listner
        (e.dispatcher as
AndroidDialog).removeEventListener(BaseEvent.COMPLETE, OnDialogClose);

        //parsing result
        switch((AndroidDialogResult)e.data) {
        case AndroidDialogResult.YES:
            Debug.Log ("Yes button pressed");
            break;
        case AndroidDialogResult.NO:
            Debug.Log ("Yes button pressed");
            break;

        }

        string result = e.data.ToString();
        AndroidNative.showMessage("Result", result + " button pressed");
    }
}

```

AndroidDialogResult result can contain: YES, NO of **AndroidDialogResult** class.

Android Message Pop Up

Creation:

```

AndroidMessage msg = AndroidMessage.Create("Message Titile", "Message
message");

```

Listeners:

```

msg.addEventListener(BaseEvent.COMPLETE, OnMessageClose);

```

onDialogClose function example:

```

private void OnMessageClose(CEvent e) {
    (e.dispatcher as
AndroidMessage).removeEventListener(BaseEvent.COMPLETE, OnMessageClose);
    AndroidNative.showMessage("Result", "Message Closed");
}

```

Example Scenes

Package contains example scene under

Assets/Extensions/AndroidNative/Example/Scenes/BillingExample

Assets/Extensions/AndroidNative/Example/Scenes/PlayServiceExample

This scene is demonstrated how to use native alert and billing

Billing use example is implemented in [BillingExample](#) and [PaymnetManagerExample](#) classes.

Play service use example is implemented in [PlayServiceExample](#) class.

Native alerts use example is implemented in [PopUpExamples](#) class.

Support

If you have any questions, problems or suggestions, please contact me anytime via
E-mail: lacost.st@gmail.com.