

# Android Native Plugin

Anyone can comment

---

[Short Overview](#)

[How to update](#)

[Billing](#)

[Setting Up.](#)

[Setting Up for Test Purchases](#)

[Signing and Uploading apk with Unity](#)

[Classes Documentation](#)

[Play Service -](#)

[Before you begin](#)

[Step 1: Check and Download](#)

[Step 2: Set up the game in the Developer Console](#)

[Step 3: Modify your code](#)

[Step 4: Test your game](#)

[Classes Documentation](#)

[Google Cloud](#)

[Setup.](#)

[Classes Documentation](#)

[Google Cloud Messaging](#)

[Native Alerts](#)

[Android Rate Pop Up](#)

[Android Dialog Pop Up](#)

[Android Message Pop Up](#)

[AdMob](#)

[Facebook and Twitter](#)

[Google Analytics](#)

[Before you Begin](#)

[Getting Started](#)

[Classes Documentation](#)

[PlayMaker Actions](#)

[Other Features](#)

[Compile and sign with Unity](#)

[Frequently Asked Questions](#)

[AdMob Questions](#)

[Any of plugin functions is not working.](#)

[Any plugin function call causes app crash](#)

[Can I use this plugin with other Android Plugins from Asset Store](#)

[Can I cut plugin functionality.](#)

[I am getting build error](#)

[How to compile androidnative.jar from eclipse project](#)

[How to get logcat log](#)

[How to integrate Android Native with Chartboost](#)

[I do not see my friends scores under circles tab of leaderboard](#)

[The item you were attempting to purchase could not be found](#)

[Example Scenes](#)

[How to Get Support](#)

## *Short 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).

# How to update

## 1. Version Notes

With every new update I make try to make plugin better. Add new features, improve stability, usability and code base structure.

When new version is available, you can find out what's new in the version and version history by pressing version number on [Asset Store Plugin Page](#):

### Android Native Plugin

Category: Scripting/Integration  
Publisher: Stan's Assets  
Rating: ★★★★★ (141)  
Price: \$30

Buy \$30.00

fully compatible with:  
IOS Native  
Mobile Social Plugin  
Google Mobile Ads SDK  
Google Play for IOS

Play Service: (Android 2.3+ required)  
\* Detailed set up instructions  
\* Google Cloud  
\* Leader-boards  
\* Achievements  
\* Retrieve top or player center scores  
\* Play Service users names and avatars  
\* Load Friend list  
Billing:  
\* Set Up instructions  
\* In-App purchases

Version: 3.4 (May 22, 2014) Size: 17.3 MB

Support E-mail Support Website Visit Publisher's Website

## 2. Avoiding conflicts

Sometimes in order to implement new feature or improve code structure I have to change some of plugin files / folder or method names.

It will be of course described in version notes. But if you simple click update in Asset Store version, you may get duplicated or conflicted files.

To avoid this, I strongly recommend to remove all plugin files from your project before update. Currently plugin parts located in:

```
Assets/Extensions/AndroidNative/  
Assets/Extensions/GooglePlayCommon/
```

```
Assets/Extensions/StansAssetsPreviewUI/  
Assets/Extensions/FlashLikeEvents/  
Assets/Plugins/Android
```

If you own another plugins with also have `GooglePlayCommon` folder (this folder is shared between few plugins in order to supply compatibility of android plugins) I also recommend update those plugins too. To avoid conflicts

### ***3. Saving Plugins settings***

Plugin setting that was specified in editor GUI earlier will be overridden. So just backup your settings data with stored in files:

```
Assets/Extensions/AndroidNative/Resources/AndroidNativeSettings  
Assets/Extensions/GooglePlayCommon/Resources/SocialSettings
```

And replace plugin files with your backup after update.

# Billing

---

## Setting 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

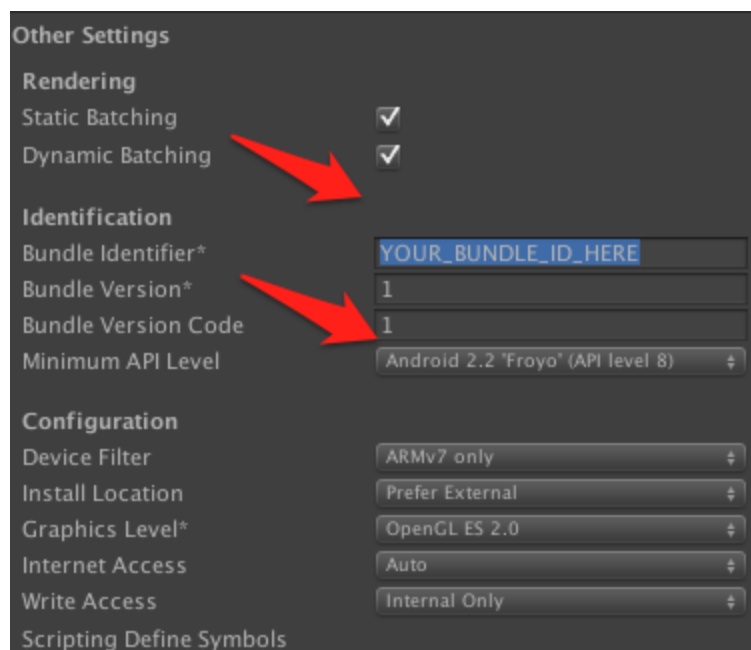
Where **example** is generally the company/publishers name, and **testapp** is the app name.

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
11     <application
12         android:icon="@drawable/app_icon"
13         android:label="@string/app_name"
14         android:debuggable="false">
15
16         <activity android:name="com.android.MainActivity" android:label="@string/app_name" and
17             <intent-filter>
18                 <action android:name="android.intent.action.MAIN" />
19                 <category android:name="android.intent.category.LAUNCHER" />
20             </intent-filter>
21         </activity>
22         <activity android:name="com.unity3d.player.VideoPlayer" android:label="@string/app_nam
23         </activity>
24
25
26     </application>
27     <uses-feature android:glEsVersion="0x00020000" />
28     <uses-sdk
29         android:minSdkVersion="8"
30         android:targetSdkVersion="15" />
31     <uses-permission android:name="android.permission.INTERNET" />
32     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
33
34     <!-- VERY IMPORTANT! Don't forget this permission, or in-app billing won't work. -->
35     <uses-permission android:name="com.android.vending.BILLING" />
36
37 </manifest>

```

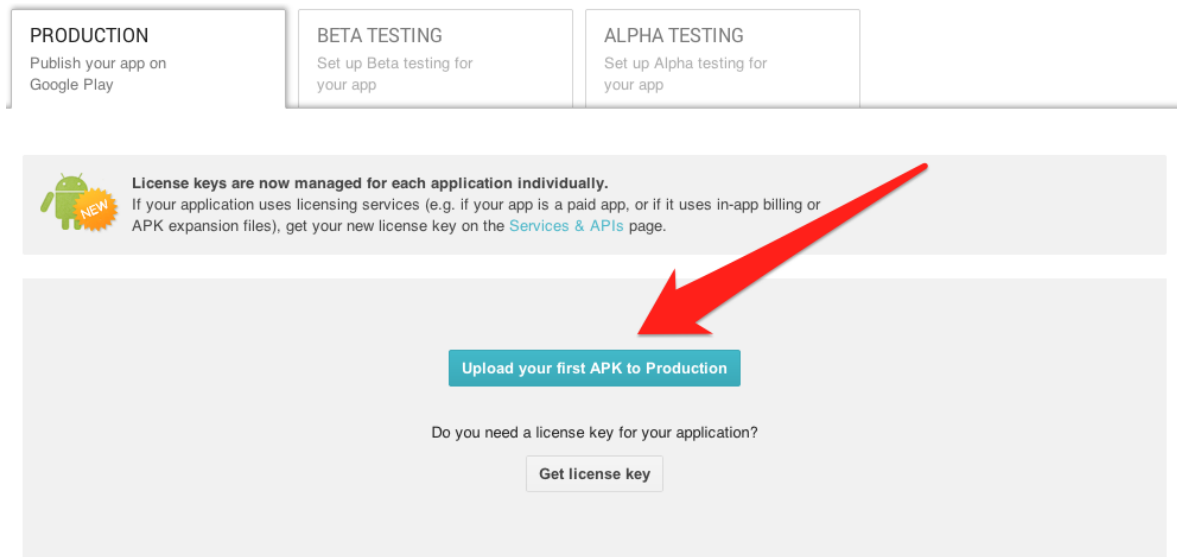


You can build your signed apk file now. Just press **build** button.

**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 example scene and try to modify [PaymnetManagerExample](#) class to work with your products, or create your own using [PaymnetManagerExample](#) as example.

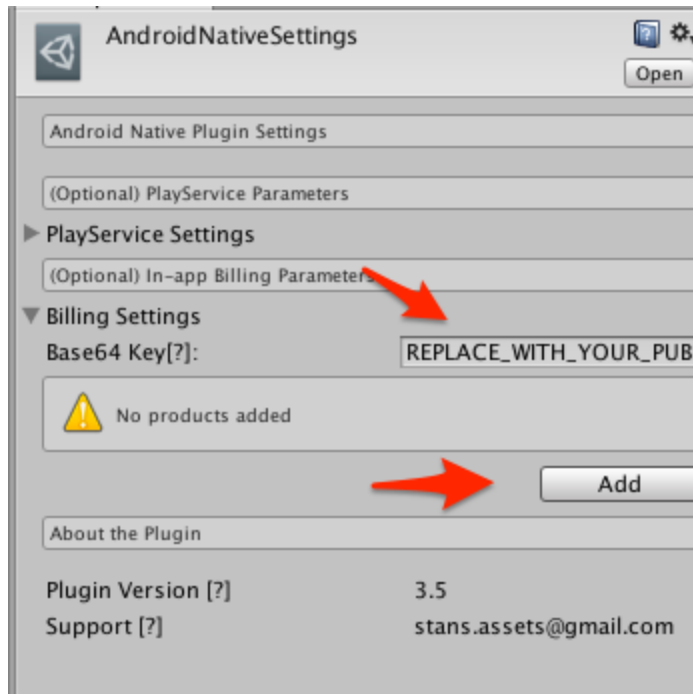
#### Note:

**Testing for In-App Billing, Licensing, and APK Expansion Files** Previously, you could use your Draft APKs to test these features. After a recent change, testing your features using Draft APKs is no longer supported. You must now publish your app to the alpha or beta distribution channels to test your implementation of these features. For more information, please refer to the documentation available on the Android Developers site.

## Fill In Plugins Billing Settings

Open Plugin Settings window:

Windows → Android Native → Edit Settings



Set your Base64 key, that you got in [Setup](#) section. And also fill product list ids for your app.

# Classes Documentation

## AndroidInAppPurchaseManager class.

---

### API methods:

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

*You can ignore this function if you already set all your product id's in [plugin setting](#)*

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

*Connecting to the Android Market with your public key. If you leave key filed empty, it will use base64 key specified in [plugin settings](#). Triggers ON\_BILLING\_SETUP\_FINISHED event*

```
public void loadStore()
```

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

*Get's registred products details. Triggers ON\_RETRIEVE\_PRODUC\_FINISHED event*

```
public void retrieveProducDetails()
```

*Purchase the product. Triggers ON\_PRODUCT\_PURCHASED event.*

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

*Consume the product. Triggers ON\_PRODUCT\_CONSUMED event.*

```
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<GooglePurchaseTemplate> purchases
```

*List of registered products*

```
public List<GoogleProductTemplate> products
```

## GoogleProductTemplate 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
```

*price in micros values*

```
public string priceAmountMicros;
```

*product currency code*

`public string priceCurrencyCode;`

## GooglePurchaseTemplate 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;`

*purchase signature*

`public string signature;`

*purchase token*

`public string token;`

*purchase time*

`public long time;`

*original unparsed google response.*

`public string originalJson;`



*purchase state*

```
public GooglePurchaseState state;
```

## 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 GooglePurchaseTemplate 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

---

## Before you begin

- You should have your Android development environment set up.
- You should have a physical device running Android 2.3 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.
  - c. Follow instructions to [generate an OAuth 2.0 client ID](#) for your Android app.
  - d. When linking your Android app, make sure to specify the exact package name you used previously when renaming sample package.
  - e. 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](#).
3. 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.

4. 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 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](#).
5. 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](#).
6. [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

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.*

*To set connection permissions open Windows->Android Native → Edit settings*

`public void connect()`

*Disconnect from Play Service*

`public void disconnect()`

### Getters:

*Current connection state*

`public static GPConnectionState state`

*True if init function was already called*

`public bool isInitialized`

### Events:

*Fires when [GooglePlayConnection](#) state is CONNECTED. Event data null;*

`PLAYER_CONNECTED`

*Fires when [GooglePlayConnection](#) state is DISCONNECTED. Event data null;*

`PLAYER_DISCONNECTED`

*Fires when connection result was received. Event data contains*

*GooglePlayConnectionResultCode*

CONNECTION\_RESULT\_RECEIVED

*Fires when connection state was changed. Event data contains*

*GPCConnectionState*

CONNECTION\_STATE\_CHANGED

GooglePlayManager : Singleton<GooglePlayManager> class.

---

### API methods:

*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 showLeaderBoardById(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 submitScoreById(string leaderboardId, int score)

*Trigger Leaderboards info request, LEADERBOARDS\_LOEADED event will be fired on complete*

public void loadLeaderBoards()



*Asynchronously load the player-centered page of scores for a given leaderboard. If the player does not have a score on this leaderboard, this call will return the top page instead.*

```
public void loadPlayerCenteredScores(string leaderboardId, GPBoardTimeSpan span,
GPCollectionType collection, int maxResults)
```

*Asynchronously load the top page of scores for a given leaderboard.*

```
public void loadTopScores(string leaderboardId, GPBoardTimeSpan span,
GPCollectionType collection, int maxResults)
```

*Trigger achievement report request, ACHIEVEMENT\_UPDATED event will be fired on complete*

```
public void reportAchievement(string achievementName)
```

```
public void reportAchievementById(string achievementId)
```

*Trigger achievement reveal request, ACHIEVEMENT\_UPDATED event will be fired on complete*

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

```
public void revealAchievementById(string achievementId)
```

*Trigger achievement increment request, ACHIEVEMENT\_UPDATED event will be fired on complete*

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

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

*Trigger achievement info load request, ACHIEVEMENTS\_LOADED event will be fired on complete*

```
public void loadAchivments()
```

*Load player connected players data (friends).*

```
public void loadConnectedPlayers()
```

**Public methods:**

*Get's Leader board by id*

```
public GPLEaderBoard GetLeaderBoard(string leaderboardId)
```

*Get's Achievement board by id*

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

*Get's player by id*

```
public GooglePlayerTemplate GetPlayerById(string playerId)
```

## Getters:

*Information about current player*

```
public GooglePlayerTemplate player
```

*Loaded players Dictionary*

```
public Dictionary<string, GooglePlayerTemplate> players
```

*Loaded Leaderboards*

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

*Loaded Achievements*

```
public Dictionary<string, GPAchievement> achievements
```

*Loaded friends ids*

```
public List<string> friendsList
```

## Events:

*Fires on Leaderboard score submitted. Event data contains GooglePlayResult.*  
**SCORE\_SUBMITTED**

*Fires on Leaderboards data loaded. Event data contains GooglePlayResult.*

## LEADERBOARDS\_LOEADED

*Fires when friends data loaded.*

## FRIENDS\_LOADED

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

## ACHIEVEMENT\_UPDATED

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

## ACHIEVEMENTS\_LOADED

*Fires when player request loaded*

## SCORE\_REQUEST\_RECEIVED

## 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 all currently loaded scores*

```
public List<GPScore> GetScoresList(GPBoardTimeSpan timeSpan, GPCollectionType collection)
```

*Get's score by player id*

```
public GPScore GetScoreByPlayerId(string playerId, GPBoardTimeSpan timeSpan, GPCollectionType collection)
```

*Get's score by rank*

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

*Get's current player score*

```
public GPScore GetCurrentPlayerScore(GPBoardTimeSpan timeSpan, GPCollectionType collection)
```

*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;`



# Google Cloud Messaging

To create a Google API project:

1. Open the [Google Cloud Console](#).
2. If you haven't created an API project yet, click **Create Project**.
3. Supply a project name and click **Create**.
4. Once the project has been created, a page appears that displays your project ID and project number. For example, **Project Number: 670330094152**.
5. Copy down your project number. You will use it later on as the [GCM sender ID](#).

## Enabling the GCM Service

---

To enable the GCM service:

1. In the sidebar on the left, select **APIs & auth**.
2. In the displayed list of APIs, turn the **Google Cloud Messaging for Android** toggle to ON.

## Obtaining an API Key

---

To obtain an API key:

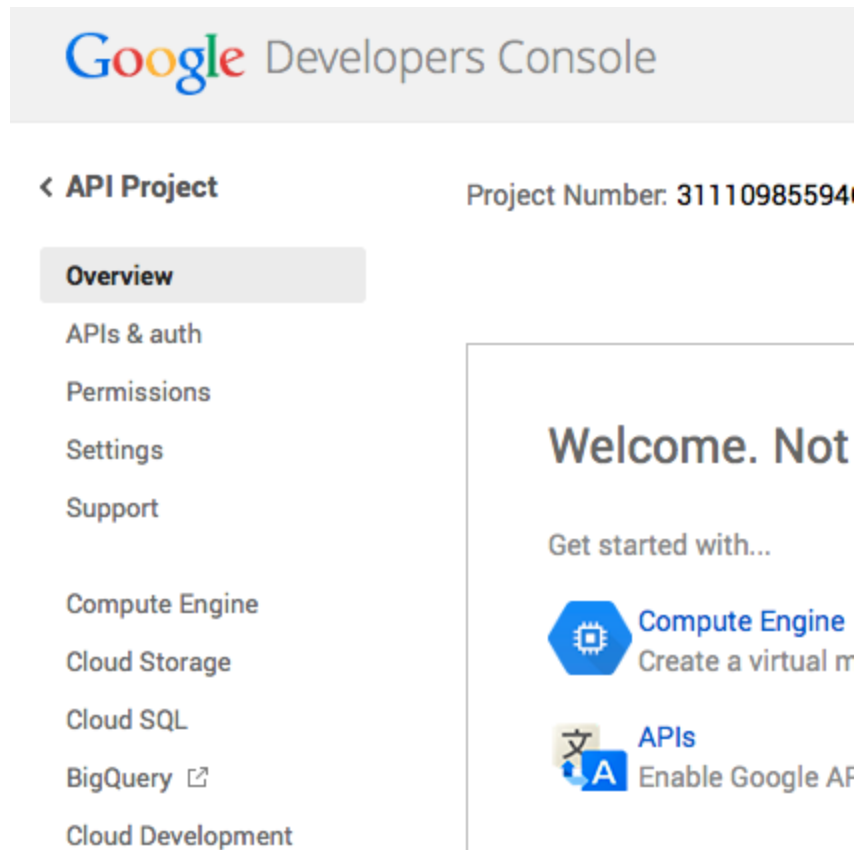
1. In the sidebar on the left, select **APIs & auth > Registered apps**.
2. Click **Register app**.
3. In the **Name** field, type your app's name.
4. Click **Android > Accessing APIs directly from Android**.
5. Under **Android identification**, type the package name for your app.
6. Enter an SHA1 fingerprint. To get this value, follow the instructions in the [console help](#).
7. Click **Register**.
8. In the new page, open the **Android Key** section and copy the [API key](#). You will need the API key later on to perform authentication in your application server.
9. **Note:** If you need to rotate the key, click the "recycle key" icon. A new key will be created. If you think the key has been compromised and you want to delete it immediately, you can accomplish this by deleting the app from the console. Then create a new entry for the app with the same SHA1 and package name.

Now lets configure Google Developer Console.

1. Open the [Google APIs Console page](#)

2. If you have never created a project before, the page will prompt you to create one

In other case, the Dashboard page is displayed for the last project previously created. You can use that project or create a new one in the *API Project -> Create Project* menu




In any case, take note of the value of **Project Number** present on the screen. This value will be used later when configuring the Smart Devices Main Object in GeneXus to enable it to receive notifications (*Sender ID* property).





3. Enable GCM Service (if it is not) in the *APIs* option of the *APIs & auth* menu.

Google Developers Console			
		+Sebastián 	
< PushNotifications	NAME	QUOTA	STATUS
Overview	<a href="#">Google Cloud Messaging for Android</a>		<input checked="" type="checkbox"/>
APIs & auth	<a href="#">Google Maps Android API v2</a>		<input checked="" type="checkbox"/>
APIs	<a href="#">Ad Exchange Buyer API</a>	1,000 requests/day	<input type="checkbox"/>
Credentials	<a href="#">Ad Exchange Seller API</a>	10,000 requests/day	<input type="checkbox"/>
Consent screen	<a href="#">Admin SDK</a>	150,000 requests/day	<input type="checkbox"/>
Push			

4. Generate the API Key. Go to *Credentials* option in the *APIs & auth* menu, select *Create new Key* and *Server Key*.

## < PushNotifications

Overview

APIs & auth

APIs

Credentials

Consent screen

Push

Permissions

Settings

Support

Compute Engine

Cloud Storage

Cloud SQL

BigQuery 

## OAuth

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private.

[Learn more](#)

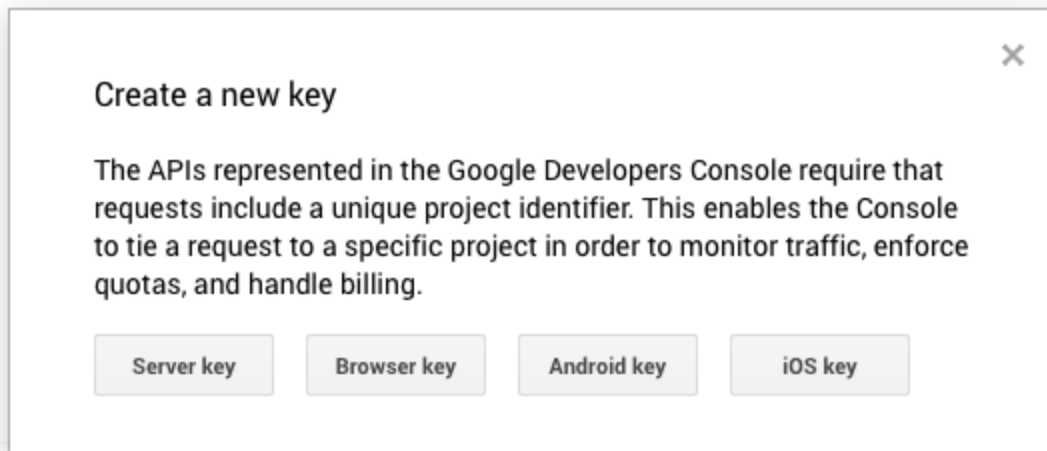
CREATE NEW CLIENT ID

## Public API access

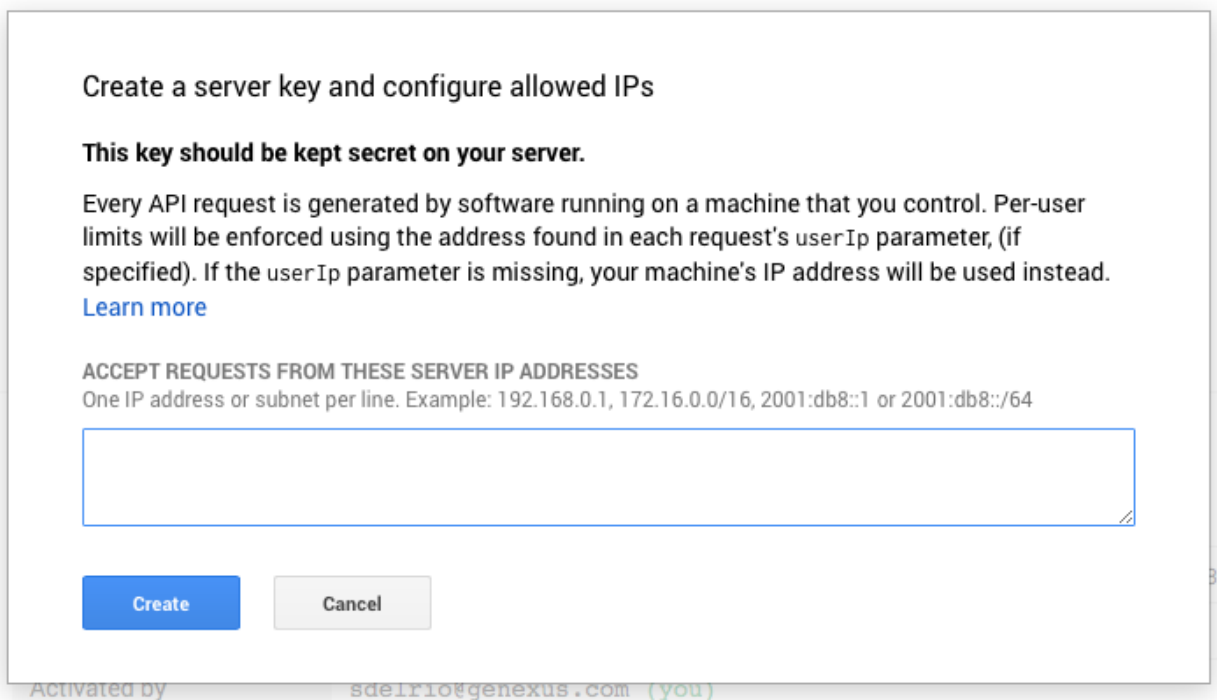
Use of this key does not require any user action or consent, does not grant access to any account information, and is not used for authorization.

[Learn more](#)

CREATE NEW KEY



In the popup window displayed you should indicate the range of IP addresses which will have permission to send notifications to this Project ID, if you want to set "All IPs" use the value 0.0.0.0/0



Now the **API Key** is finally created. You have to take note of its value in order to set the *Sender API Key* property of the Smart Devices Main Object in GeneXus, which will receive the notifications.

### Key for server applications

API key	AIzaSyAz8p02lAI-fvr dxOuDEQwbruqYYwDt2EM
IPs	0.0.0.0/0
Activation date	May 8, 2014 6:54 AM
Activated by	sdelrio@genexus.com (you)

Edit allowed IPs

Regenerate key

Delete

That is all. We are now ready to use Push Notifications.

Example of server code can be found at:

**Assets/Extensions/AndroidNative/Addons/GCMServer**

Example scene can be found at:

**Assets/Extensions/AndroidNative/xExample/Scenes/OtherFeatures**

To get the device **registration Id** id you should call

GoogleCloudMessageService.instance.RegisterDevice([SENDER\\_ID](#));

It will trigger **CLOUD\_MESSAGE\_SERVICE\_REGISTRATION\_RECIVED** or **CLOUD\_MESSAGE\_SERVICE\_REGISTRATION\_FAILED** events.

After this you can send this **registration Id** to your backend.

To find out if the push notification was received, you can use

`GoogleCloudMessageService.instance.LoadLastMessage();`

it will trigger **CLOUD\_MESSAGE\_LOADED** event. And you can get message using getter:

`GoogleCloudMessageService.instance.lastMessage`

# Native Alerts

---

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

## Android Rate Pop Up

Pop up creation:

```
AndroidRateUsPopUp 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");
}
```

# AdMob

---

Usage and Setup of AdMob is fully described in [AdMob Documentation](#).

The only difference, you should use **AndroidAdMobController** instead **GoogleMobileAd**(*crossplatform*) class.

If you own both plugin, the of course use **GoogleMobileAd** class.

# Facebook and Twitter

---

Usage and Setup of Facebook and Twitter is fully described in [Mobile Social Plugin Documentation](#)

The only difference, you should use **AndroidTwitterManager.instance** instead **SPTwitter.instance**(*crossplatform*) class.

If you own both plugins, then of course use **SPTwitter** class.

# Google Analytics

## ***Before you Begin***

Before implementing the SDK, make sure you have the following:

- An Android app that you can use to implement the Google Analytics
- A new Google Analytics [app property and view \(profile\)](#).

## ***Getting Started***

Replace tracking id in the **analytics.xml** in your project's witch located under **Assets/Plugins/Android/res/values analytics.xml**

In the Google Analytics SDK for Android v2.x and higher, you can configure your EasyTracker implementation using parameters defined in your **analytics.xml** file. The table below lists all available parameters you can use for version 2 or higher of the Google Analytics SDK for Android. Table can be founded [here](#)

You can also get more information from Google [Getting Started guide](#).

## ***Classes Documentation***

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

---

### **API methods:**

*After calling this function your app will start sending analytics to google, and will appear shortly in your report. But you can do much more with other api methods described below.*

```
public void StartTracking()
```

*Initialize a tracker using a Google Analytics property ID. By default trackingID id will be one you spesifayed in analytics.xml but can always be changed with this function.*

```
public void SetTrackerID(string trackingID)
```

*Send the scre view*

```
public void SendView(string appScreen)
```

*Send event*

```
public void SendEvent(string category, string action, string label)
```

```
public void SendEvent(string category, string action, string label, long value)
```

```
public void SendEvent(string category, string action, string label, string key, string val)
```

*Send timing event*

```
public void SendTiming(string category, long intervalInMilliseconds)
```

```
public void SendTiming(string category, long intervalInMilliseconds, string name)
```

```
public void SendTiming(string category, long intervalInMilliseconds, string name, string label)
```

*Create transaction*

```
public void CreateTransaction(string transactionId, string affiliation, float revenue, float tax, float shipping, string currencyCode)
```

*Create item for transaction.*

```
public void CreateItem(string transactionId, string name, string sku, string category, float price, int quantity, string currencyCode)
```



*Set session key*

```
public void SetKey(string key, string value)
```

*Remove Session key*

```
public void ClearKey(string key)
```

*Set Log Level*

```
public void SetLogLevel(int lvl)
```

*It you will set dry run as **true**, analytics will not be sent to google.  
Function for testing purposes.*

```
public void SetDryRun(bool mode)
```

# Other Feature

*Activate Immersive Mode (Available from Android 4.4)*

```
ImmersiveMode.instance.EnableImmersiveMode();
```

*Show Native Preload*

```
AndroidNativeUtility.ShowPreloader();
```

*Hide Native Preload*

```
AndroidNativeUtility.HidePreloader();
```

*Share the message using ACTION\_SEND*

*first param is dialog window title, second is your message*

```
AndroidSocialGate.StartShareIntent("Hello Share Intent", "This is my text to share");
```

*Share the message with Texture2D using ACTION\_SEND*

```
AndroidSocialGate.StartShareIntent("Hello Share Intent", "Sharing Hello wolrd image",  
shareTexture);
```

*Share message with filters. For example if you want to share image by*

**Mail:**

```
AndroidSocialGate.StartShareIntent("Hello Share Intent", "Sharing Hello wolrd image",  
shareTexture, "mail");
```

**Twitter:**

```
AndroidSocialGate.StartShareIntent("Hello Share Intent", "This is my text to share",  
shareTexture, "twi");
```

**Facebook:**

```
AndroidSocialGate.StartShareIntent("Hello Share Intent", "This is my text to share",  
tex, "face");
```

**Note:** Facebook does not allow you to share the text, it will be ignored, due to Facebook policies.



# Compile and sign with Unity

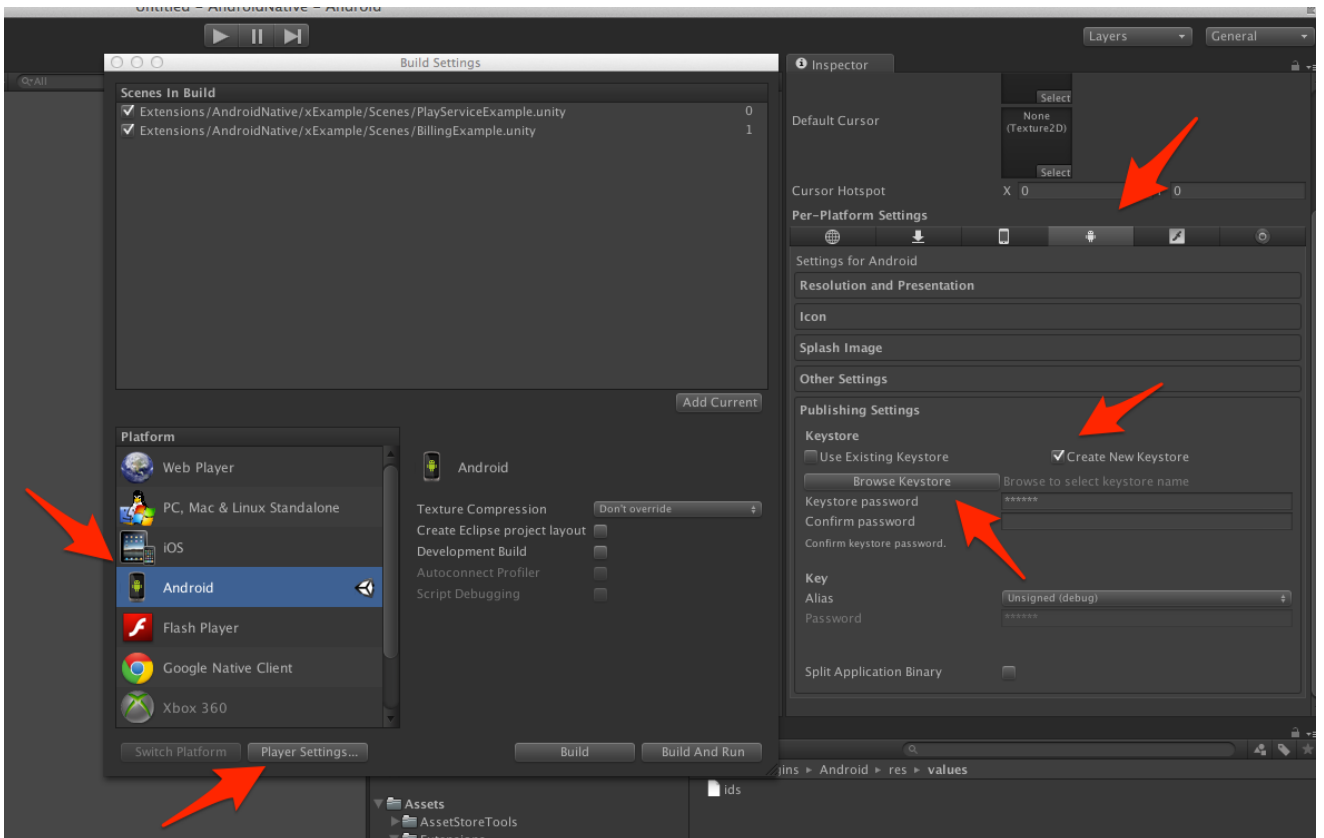
Go to:

**File → Build Settings**

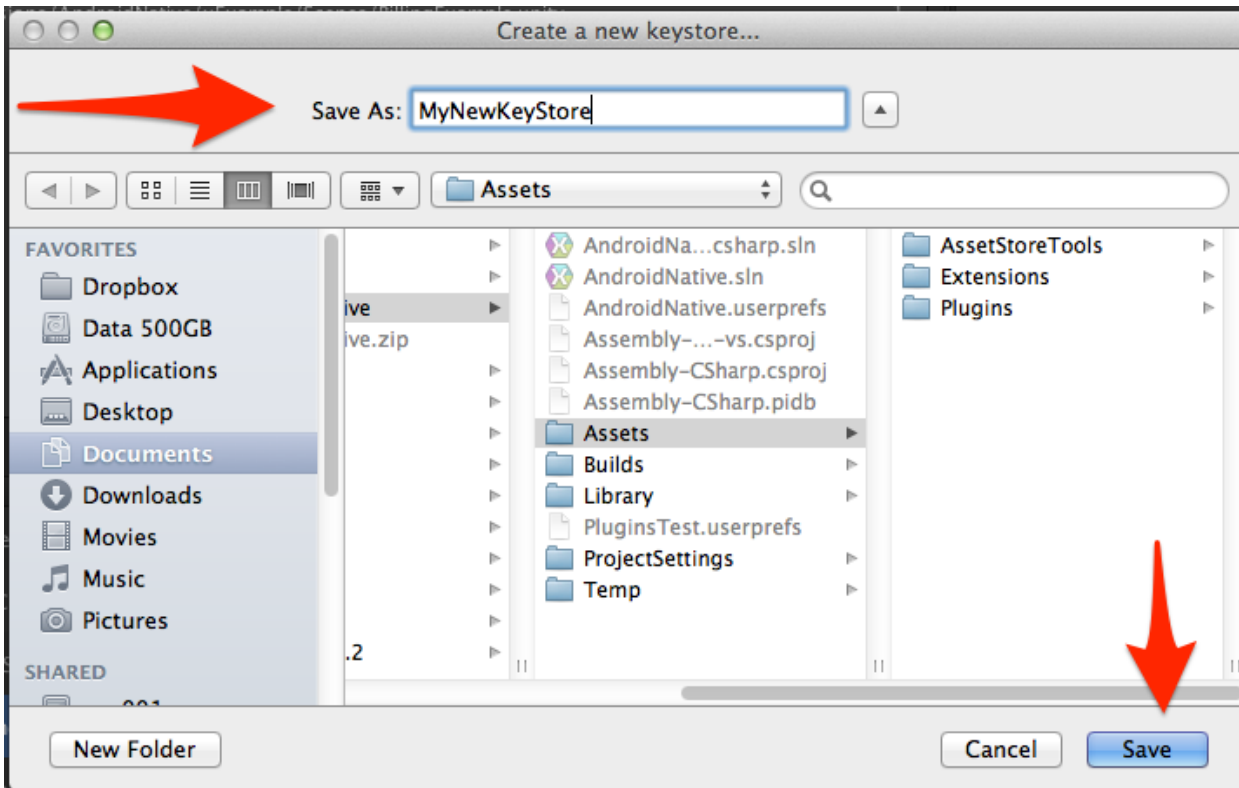
Choose Android platform and press **Player Settings** button.

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.



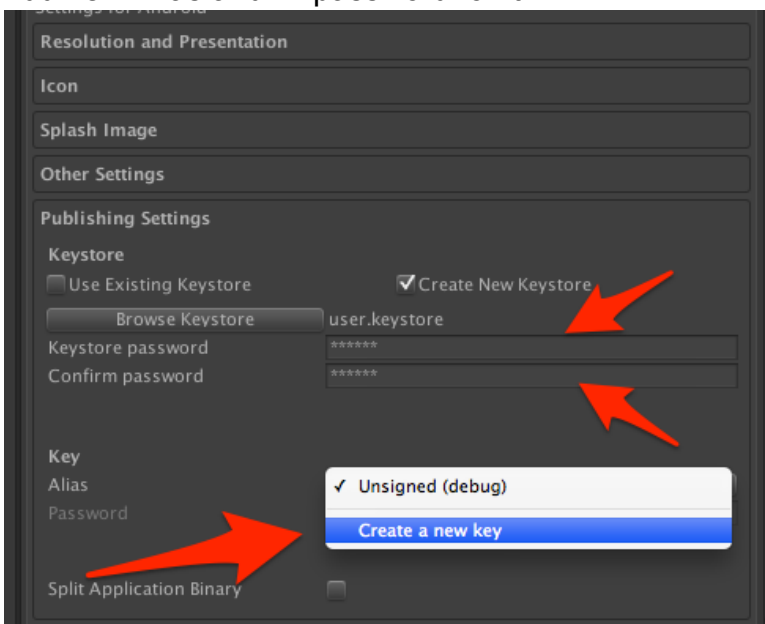
in the dialog box, select the path and name for the new keystore.



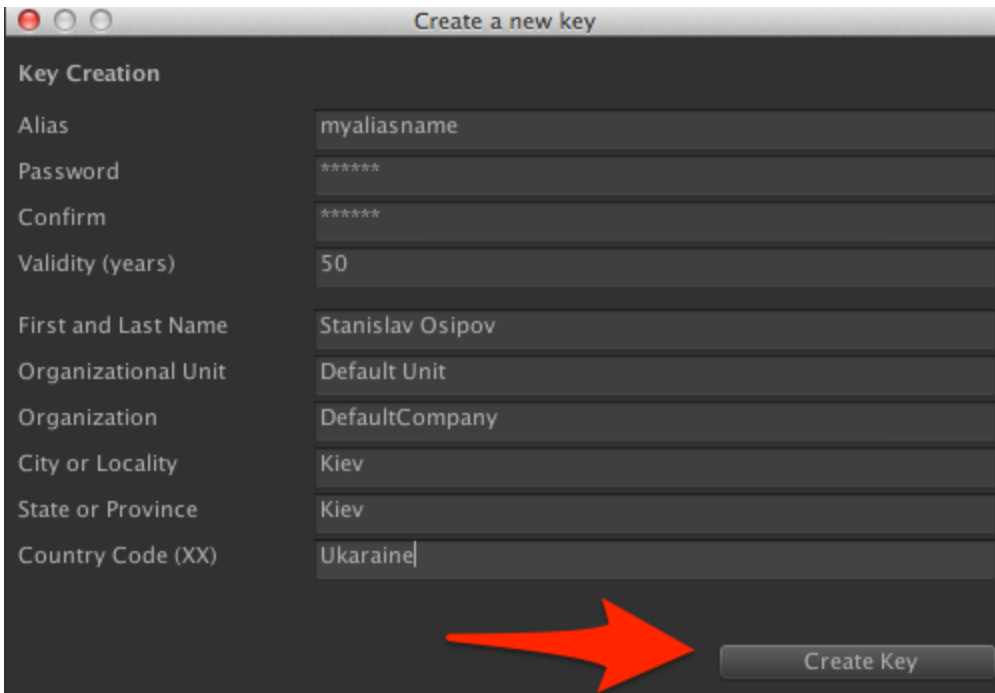
And press “**Save**” button.


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](#).

[Lean 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**.

# *PlayMaker Actions*

---

The plugin now contains playmaker actions.

The actions scripts can be found in the zip archive at:

## **Assets/Extensions/AndroidNative/Addons/PlayMakerActions**

You can simply unrar it to the same folder and Android Native actions will appear under playmaker actions menu. You always welcome on the [PlayMaker Actions Forum Thread](#) to request new actions or report a bug.

The current actions list is:

### **Billing**

- AN\_initBilling
- AN\_Purchase
- AN\_Consume
- AN\_PurchaseAndConsume

### **PlayService**

- AN\_PlayServiceinit
- AN\_ReportAchievement
- AN\_ShowAchivmentsUI
- AN\_ShowLeaderboards
- AN\_ShowLeaderboardUI
- AN\_SubmitScore

### **Native PopUps**

- AN\_DialogPopup
- AN\_MessagePopup
- AN\_RatePopup
- AN\_ShowPreloader
- AN\_HidePreloader

### **Google Mobile Ad**

- AN\_InitGoogleAd
- AN\_SetAdTargeting
- AN\_SetAdTestDevices



- AN\_CreateBanner
- AN\_DestroyBanner
- AN\_HideBanner
- AN\_ShowBanner
- AN\_RefreshBanner
- AN\_StartInterstitialAd
- AN\_LoadInterstitialAd
- AN\_ShowInterstitialAd

# Frequently Asked Questions

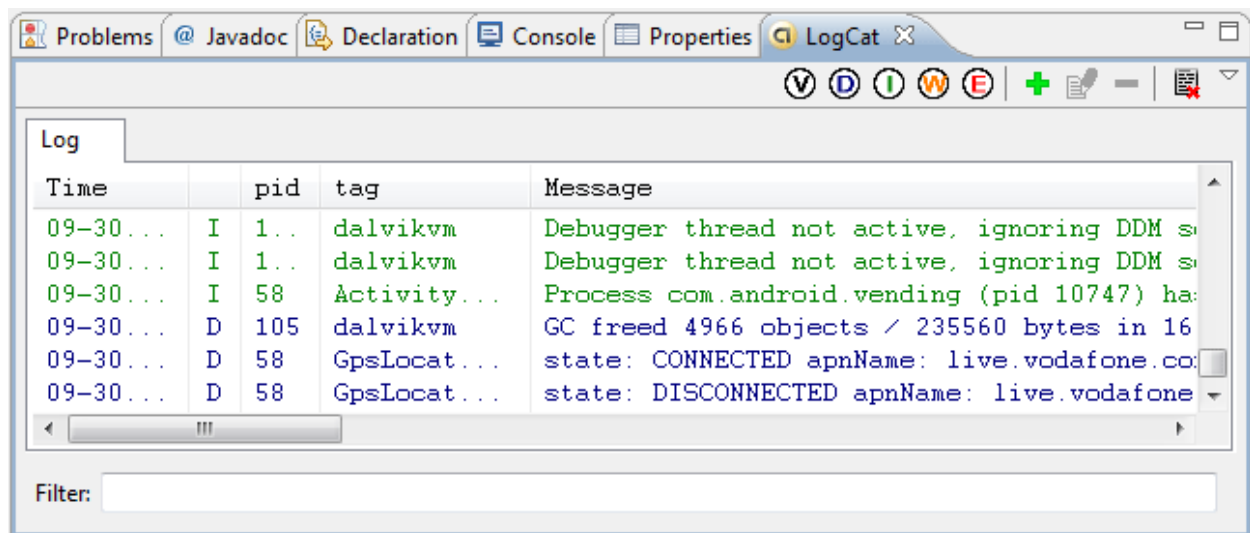
---

## ***Any of plugin functions is not working.***

Plugin will work only on **real device**, do not try to use in in the Unity Editor all plugin function calls will be simply ignored.

## ***Any plugin function call causes app crash***

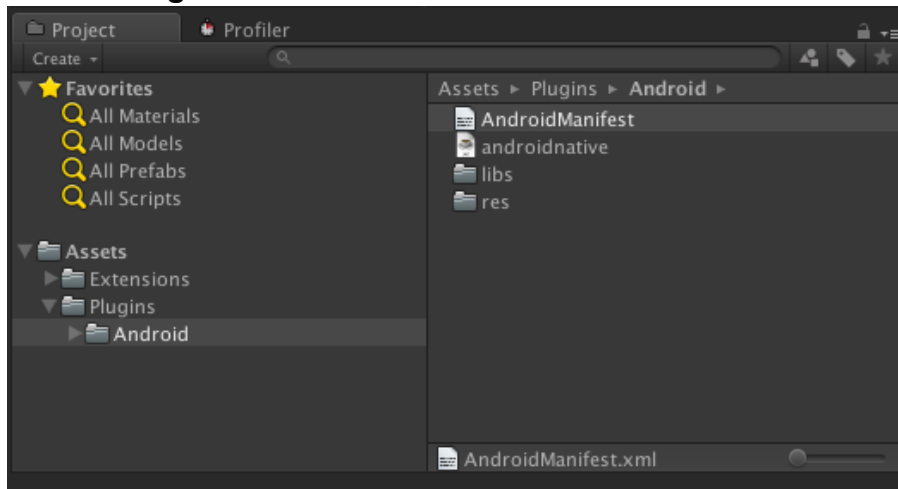
Something wrong with plugin setup. Please open Eclipse Log Cat window (or any other android logger you got) and search the exception stack trace with causes android application crash.



It stack trace contains something like **ClassNotFoundException**: **com.androidnative.AndroidNativeBridge** that plugin can not work because plugin class is simply missing in the build.

Solution:

1. Make sure that you have **androidnative.jat** and **AndroidManifest.xml** under your **Assets/Plugins/Android** folder.



2. Make sure that **AndroidManifest.xml** contains lines:

```
<activity android:name="com.androidnative.AndroidNativeBridge"
android:label="@string/app_name" ...>
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

### ***Can I use this plugin with other Android Plugins from Asset Store***

Yes you can. But that is not so easy like with IOS plugins for example.

However I can not give straight forward instruction on how to integrate my plugin with any other.

**Here is reasons why:**

- 1) I can not keep track of other plugins changes
- 2) My plugins is also may have changes and I do not know how it would affect other plugins
- 3) Not all plugins have open source

**What can I do**

- 1) Provide open source Eclipse project with clean coding
- 2) Give general instruction how to combine two plugins (can be found below)

When you build Unity app for android without any plugins, main application activity class is **UnityPlayerActivity**.

When you using Android Native Plugin it replace **UnityPlayerActivity** class by **AndroidNativeBridge** class with is extended from **UnityPlayerActivity**.

*Without plugin:*

Android App → UnityPlayerActivity

*With plugin*

Android App → AndroidNativeBridge → UnityPlayerActivity

With mean is you want to use 2 plugin in you project you have to extend one plugin from another. To have picture like:

Android App → AndroidNativeBridge → OtherPlugin → UnityPlayerActivity

or:

Android App → OtherPlugin → AndroidNativeBridge → UnityPlayerActivity

To be able to do this you should have at least one plugin with full open source and source eclipse project. **Android Native Plugin** comes with full open source and eclipse source project.

```
2
3+ import java.util.ArrayList;
34
35 public class AndroidNativeBridge extends UnityPlayerActivity {
36
```

For example you have another plugin you want to use with **Android Native Plugin**.

- Open Android Native Eclipse project.
- Add Other Plugin jar file to the project
- Extend **AndroidNativeBridge** from other plugin Activity class.
- Rebuild **androidnative.jar** and replace it in your project

After this step both plugin should work correctly.

## ***Can I cut plugin functionality.***

Some developers wish to keep their project as clean as possible and do not want to keep unused assets or code in the project.

And that is a common question how to remove some of the plugin parts.

I am developing plugins as one complete project, so I can not give straight forward instructions how to delete some features. Besides I would not recommend to do that. Cutting few scripts from the project will save you couple of bytes, but you will lose useful feature with you will probably want to use in the future and you also may harm the whole plugin by doing this.

But actually there is a reason to cut Facebook part if you not using this. Because it uses Unity Official Facebook Plugins, which will add around 5MB to your final build. To cut the facebook you should delete following folders and files:

*Assets/Facebook*

*Plugins/Android/facebook*

*Assets/Extensions/GooglePlayCommon/Social/Facebook*

*Assets/Extensions/AndroidNative/xExample/FacebookAndroidUseExample.cs*

If you still want to cut other feature, that you can do this at your own risk, Full plugins source is opened including eclipse project.

## ***I am getting build error***

if your exception looks similar to this:

```
Error building Player: Win32Exception: ApplicationName='C:\Program Files
(x86)\Java\jre6\bin\javac.exe', CommandLine='-bootclasspath
"C:/adt-bundle-windows-x86_64-20131030/sdk/platforms/android-19\android.jar" -d
"C:\Company\Games\AdTesting\AdSense2\Temp\StagingArea\bin\classes" -source 1.6 -target 1.6
-encoding ascii "com\facebook\android\Manifest.java" "com\facebook\android\R.java"
"com\SplitArrowStudios\AdSense2\Manifest.java" "com\SplitArrowStudios\AdSense2\R.java"',
CurrentDirectory='C:\Company\Games\AdTesting\AdSense2\Temp\StagingArea\gen'
```

That is common problem with Unity Official Facebook SDK (with is part of my

compatibility platform)

So basically you have 2 options to fix it

1) Instal 32 bit java.

or

1) Remove Facebook plugin part. You can find out [here](#) how to do this

## *How to compile androidnative.jar from eclipse project*

First of all we need to unrar the project.

Project archive can be found at:

**Asstes/Extensions/GooglePlayCommon/Eclipse/ANEclipseProject.rar**

You can replace this rar file to any comfortable place for your eclipse project and unrar in there.

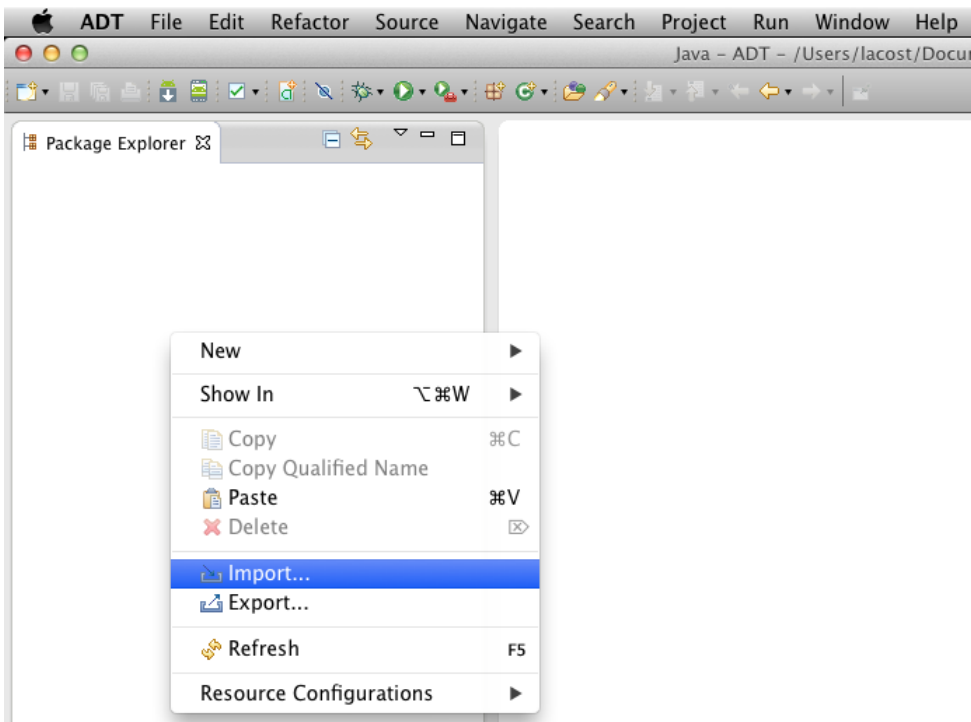
You will find two folders in rar archive. **AndroidNativeEclipce** project and **facebook** project for Unity Official SDK

Name	▲	Date Modified	Size
.DS_Store		Yesterday, 8:53 PM	6 KB
▶ AndroidNativeEclipce		Apr 22, 2014, 12:11 PM	--
▶ facebook		Apr 22, 2014, 12:11 PM	--

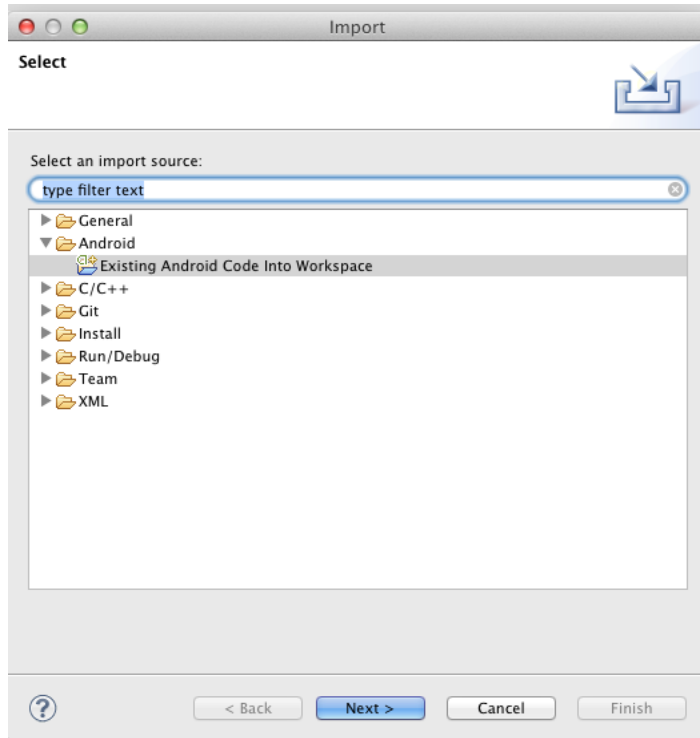
Now we need to import those project to ADT. Run the android ADT (Eclipse) you can get it from [Android SDK download page](#).



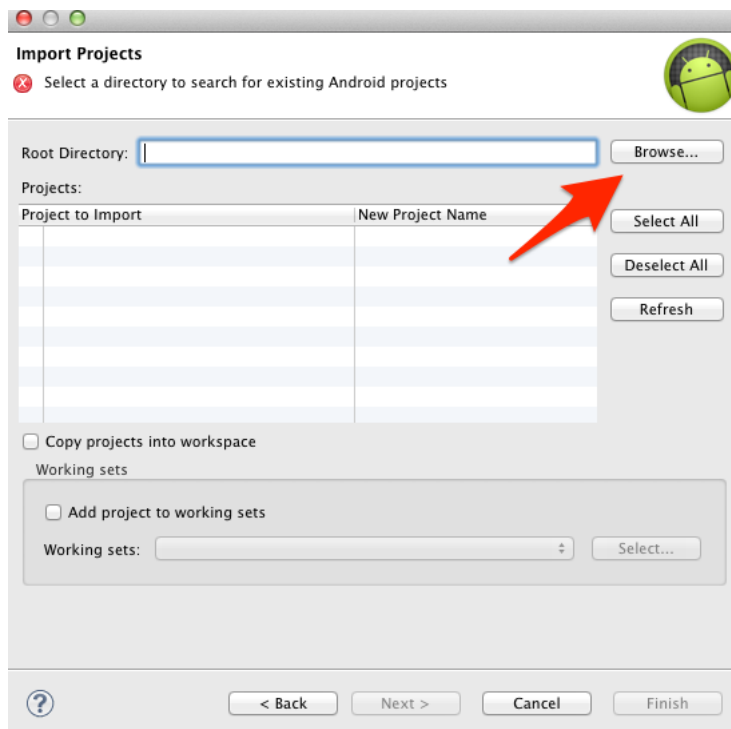
After ADT is launched right click on **Package Explorer** tab and choose **Import** menu item.



In the import setting choose **Android** → **Existing Android Code Into Workspace**

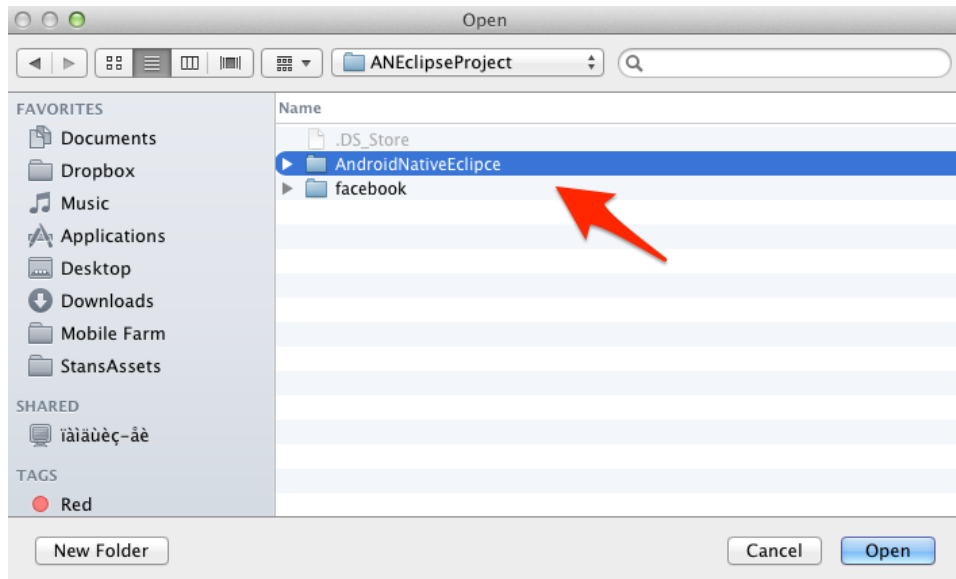


Hit next and press browse button.

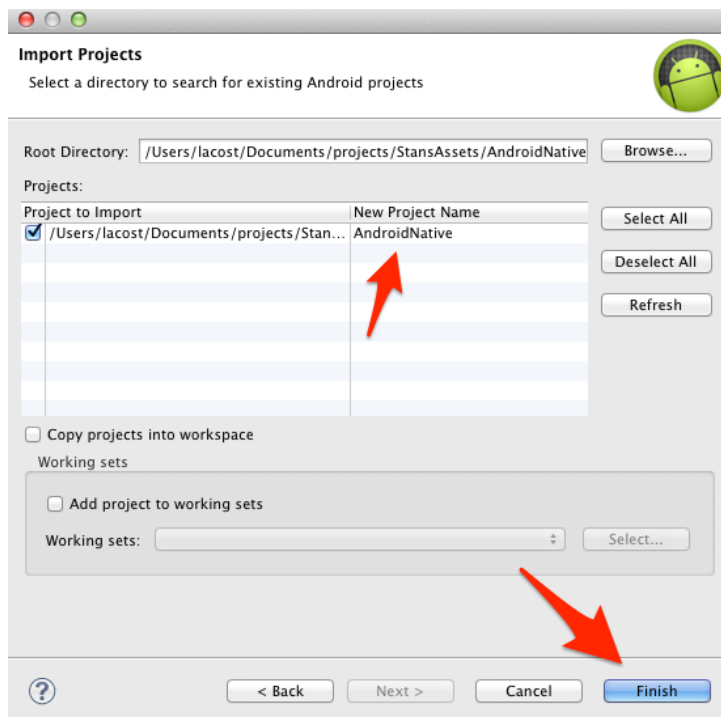


Browse to the **AndroidNativeEclipse** project location and choose **AndroidNativeEclipse** folder and press **Open**.



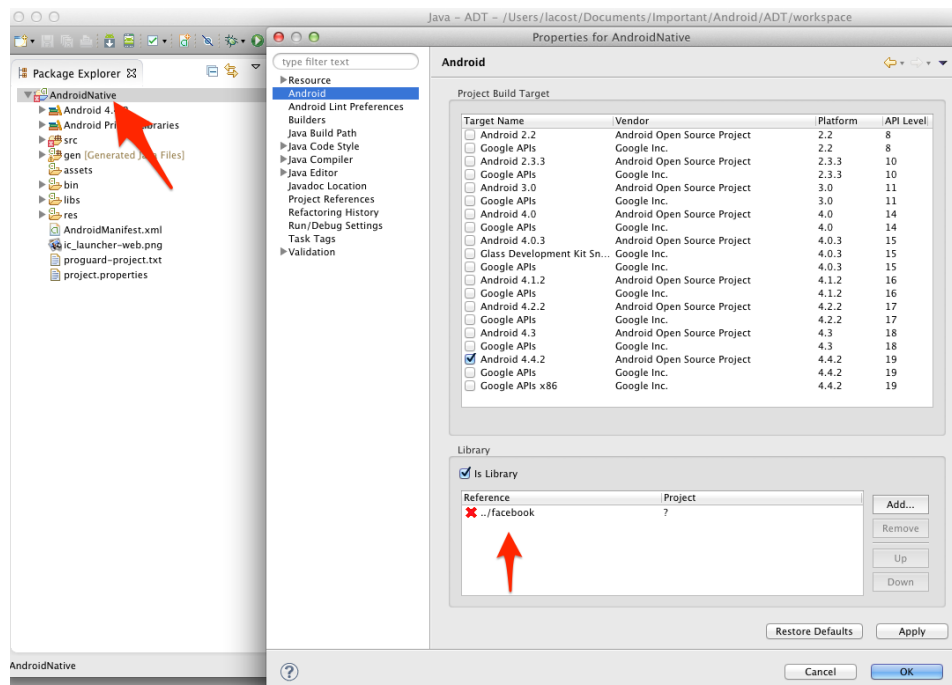


Then just press **Finish** button.



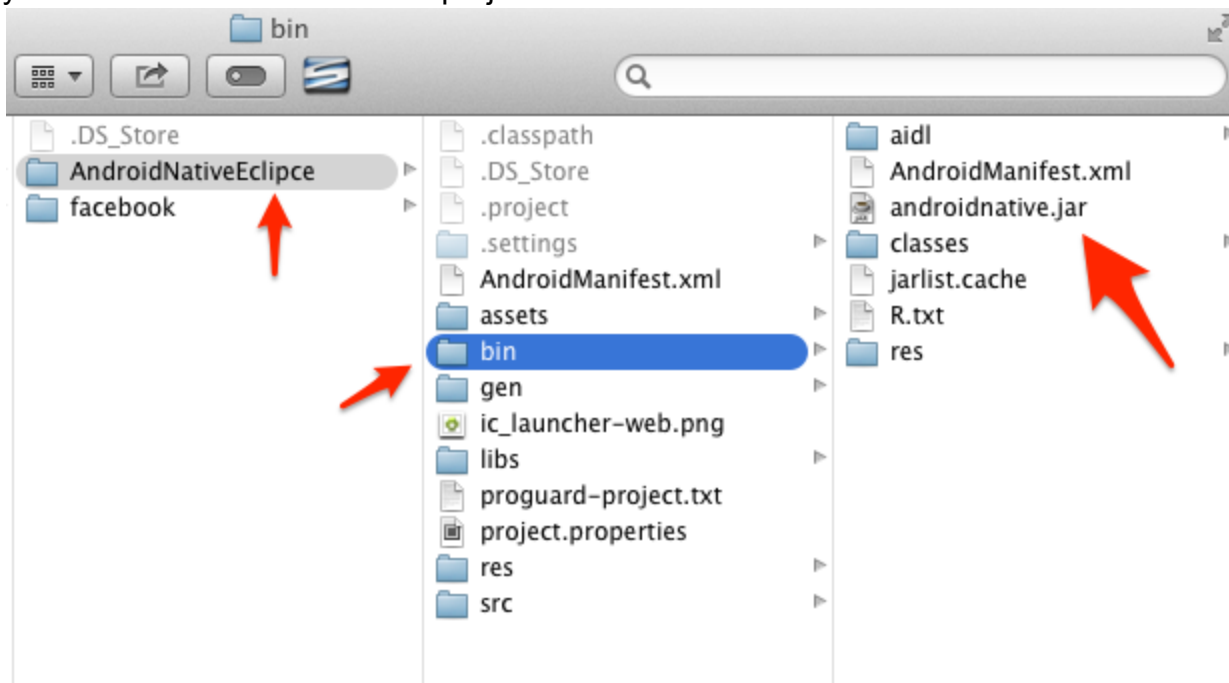
You will see **AndroidNative** project imported to the ADT. It will have compilation errors because if we will open the Project Setting tab we will see that it dependent from facebook project. To get rid of those error we should import **facebook** project in the same

way we did with **AndroidNativeEclipse** project



As soon as we will do those, errors will gnone, with means we can recompile **androidnative.jar**.

When you will change any script in the project jar file will be recompiled automatically, and you be able to find it under the project **bin** folder.



That it, now you can add your stuff and replace **androidnative.jar** in your Unity project

## How to get logcat log

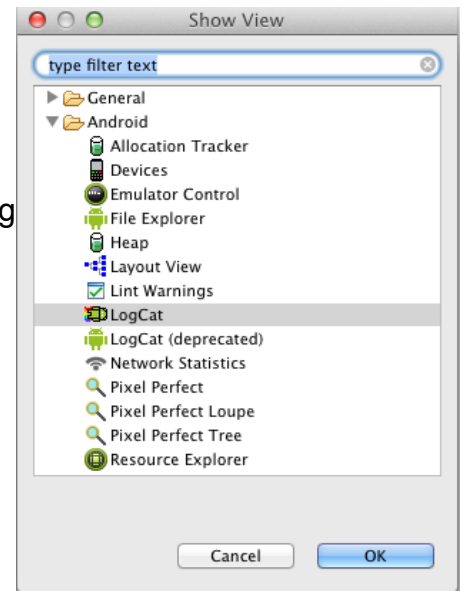
For getting the logcat log you can use should:

1. Enable USB debugging in your device.
2. Connect device to computer
3. Use console command `$ adb -d logcat`

Instead of console command you may use any other visual log viewers for android. For example from the ADT(Eclipse) with you got [Android SDK download page](#).

To do this open ADT, choose **Window** → **Show View** → **Other...**

It will open Show View window. Choose **Android** → **LogCat**  
And you will able to see the logs from your device.



## How to integrate Android Native with ChartBoost

First off all read [this part](#) of documentation. So I decided to create small tutorial how you can do this.

**Warning:** You should understand that this is only general idea, after some **AndroidNative** or **Chartboost** update provided step will can be wrong and you will have to improvise.

So first of all, we need to add into **AndroidManifest.xml** you got with AndroidNative required things from AndroidManifest from ChartBoost. Currently we only need to add one more activity to **AndroidManifest.xml**

```
<activity android:name="com.chartboost.sdk.CBImpressionActivity"
    android:excludeFromRecents="true"
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />
```

And few more permissions:

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

After this we need to combine ChartBoost and AndroidNative int one jar file. However we will append AndroidNativeBridge.java class function instead of us extending.

We need to add ChartBoost jars into the **lib** folder of our project (in Eclipse and int Unity) and append following functions with the following code.

**Function:**

```
@Override  
public void onDestroy()
```

**Append with**

```
Chartboost.sharedChartboost().onDestroy(this);
```

**Function:**

```
@Override  
public void onStart()
```

**Append with**

```
Chartboost.sharedChartboost().onStart(this);  
Chartboost.sharedChartboost().startSession();
```

**Function:**

```
@Override  
public void onStop()
```

**Append with**

```
Chartboost.sharedChartboost().onStop(this);
```

Now you can recompile **androidnative.jar** and test your integration

## *I do not see my friends scores under circles tab of leaderboard*

I'm assuming that you're still in the **testing** phase and haven't actually **published** your game via the Google Play Developer Console. **Publishing is the key**. There are two tiny sentences buried in a NOTE on this Google developer page:

[https://developers.google.com/games/services/common/concepts/leaderboards#creating\\_a\\_leaderboard](https://developers.google.com/games/services/common/concepts/leaderboards#creating_a_leaderboard)

Note: Social leaderboards will initially be empty until you publish the corresponding leaderboard by using the Google Play Developer Console

Social leaderboards won't be useful until **after** you publish. i.e. You'll never see social leaderboards during testing.

***Billing stopped working. “The item you were attempting to purchase could not be found” Android in-app billing***

Here is answer from Google Support:

Thank you for contacting Google Play Developer Support and reporting the behavior you're seeing with in-app billing.

We recently made some changes to our systems and we are now requiring an app to be published before testing. We are currently recommending to publish your APK to the Alpha channel in order to test licensing, in-app billing, and expansion files. There is no need to create a special testing group in the Alpha channel to test these features, however the app must be published and not in draft mode.

We apologize for the inconvenience and are working to update our documentation to reflect these changes.

# *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.