Android Native Plugin

Anyone can comment

Short Overview

Setup

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

Game Gifting in Android

- Sending a request
- Launching your game from a request notification
- Notifying about pending request
- Handling requests using the default inbox UI
- Handling requests programmatically
- 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
- How to merge manifest with another android plugin
- 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).

Setup

All you have to do is to copy files from

Assets/Plugins/StansAssets to → **Assets/Plugins**

If you already have some files under **Assets/Plugins/Android** it mean that you may already have some android plugin installed. In this case, please read this sections with will help you to merge the plugins:

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

How to compile androidnative.jar from eclipse project

How to merge manifest with another android plugin

If the plugin has conflict with the IOS / WP8 / Android (with not overrides main activity) Please contact support team.

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](#):

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.

Warning: Check the Version Notes before update. If version notes contains

Code Refactor: version section. It means that plugin structure is changed. Some files can be removed or function names changed. If files was removed, you should remove it by your self from the project. Example or Remove notes:

Removed:
Assets/Extensions/AndroidNative/Other/Twitter
All Scene under xExample/Scenes Moved to corresponding folder
only Preview Scene should be under xExample/Scenes

Note: 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
Assets/Facebook/Resources/FacebookSettings
```

And replace plugin files with your backup after update. Or uncheck this files when you installing the 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. Previously, you could publish a "draft" version of your app for testing. This functionality is no longer supported. Instead, there are two ways you can test how a pre-release app functions on the Google Play store:

    - You can publish an app to the alpha or beta distribution channels. This makes the app available on the Google Play store, but only to the testers you put on a "whitelist".
    - In a few cases, you can test Google Play functionality with an unpublished app. For example, you can test an unpublished app's in-app billing support by using static responses, special reserved product IDs that always return a

specific result (like "purchased" or "refunded").

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

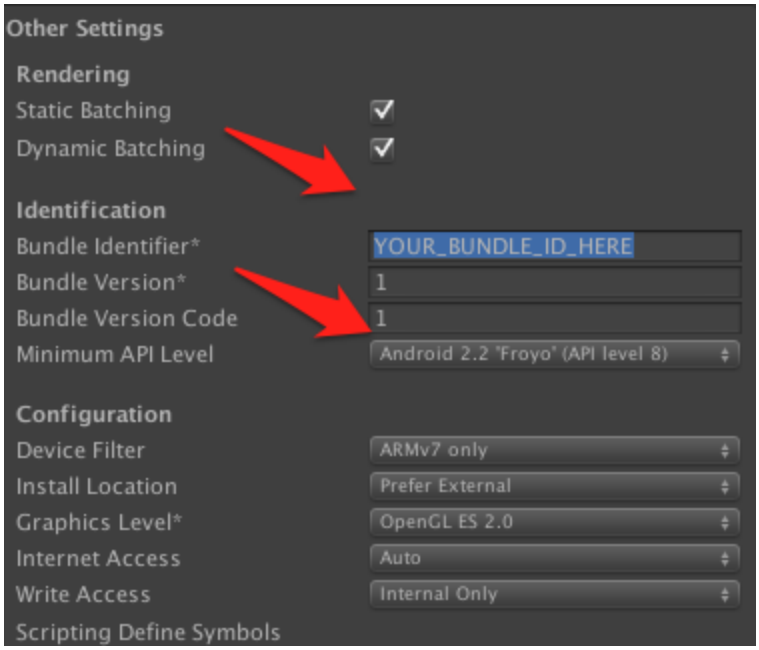
Where **example** is generally the company/publishers name, and **testapp** is the app name.

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 you 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 lasted 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

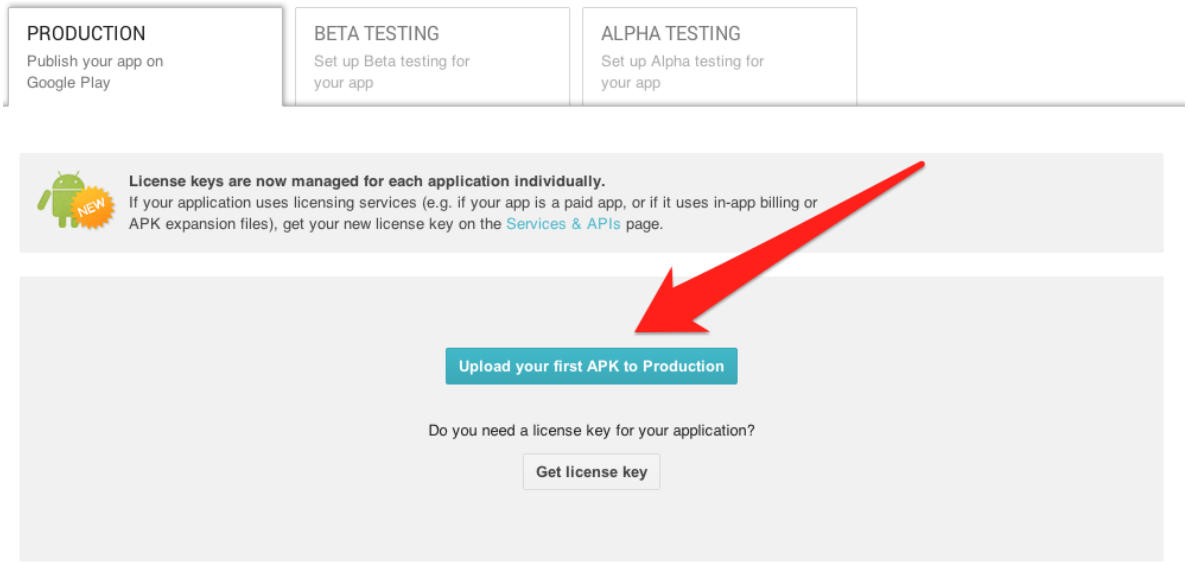
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:installLocation="preferExternal"
    package="YOUR_BUNDLE_ID_HERE"
    android:versionName="2.0"
    android:versionCode="2">

  <supports-screens android:smallScreens="true" android:normalScreens="true" android:large

  <application
      android:icon="@drawable/app_icon"
      android:label="@string/app_name"
      android:debuggable="false">

    <activity android:name="com.android.MainActivity" android:label="@string/app_name" and
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name="com.unity3d.player.VideoPlayer" android:label="@string/app_nam
    </activity>



  </application>
  <uses-feature android:glEsVersion="0x00020000" />
  <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

     <!-- VERY IMPORTANT! Don't forget this permission, or in-app billing won't work. -->
    <uses-permission android:name="com.android.vending.BILLING" />

</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.

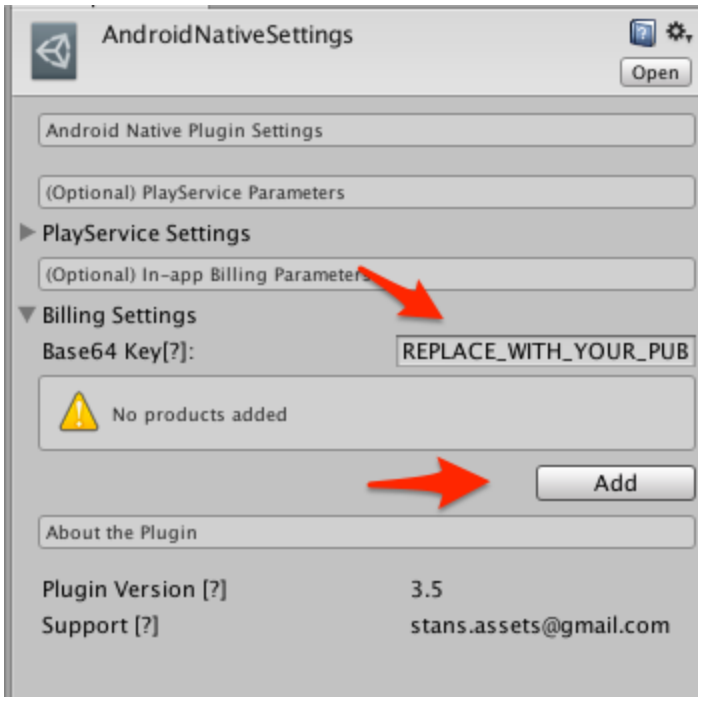
Note:

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 BILLINGHELPERR_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 instruction to add your game to the Developer Console.
    a. Follow thesWhen 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. e 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 GooglePlayConnectionResult

CONNECTION_RESULT_RECEIVED

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

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_SUBMITED 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()

Send Gift Request with Play Service UI
public void SendGiftRequest(GPGameRequestType type, int requestLifetimeDays,
Texture2D icon, string description, string playload = "")

Show Requests Inbox Dialog
public void ShowRequestsAccepDialog()

Accept Requests by id's
public void AcceptRequests(params string[] ids)

Dismiss Requests by id's
public void DismissRequest(params string[] ids)
```

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)

Get's game requst by id
public GPGameRequest GetGameRequestById(string id)
```

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

Retrieve Pending requests list friends ids
public List<GPGameRequest> gameRequests
```

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

Fires send gift result UI is received. Event data contains *GooglePlayGiftRequestResult*

SEND_GIFT_RESULT_RECEIVED

Fires when requests inbox window is dismissed. Event data is empty.

REQUESTS_INBOX_DIALOG_DISMISSED

Fires when new pending requests is detected. Event data contains *List<GPGameRequest>*

PENDING_GAME_REQUESTS_DETECTED

Fires when game request is accepted. Event data contains List<GPGameRequest>

GAME_REQUESTS_ACCEPTED

GooglePlayResult class.

Getters:

contains response result code

public GooglePlayResponceCode 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 playerId

player name

public string name

true if player has icon image

public bool hasIconImage

true if player has hi res image

public bool hasHiResImage

url of player icon image

public string iconImageUrl

public string hiResImageUrl

public Texture2D icon

public Texture2D image

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 clases

```
enum {
    TIME_SPAN_DAILY,
```

```
    TIME_SPAN_WEEKLY,
    TIME_SPAN_ALL_TIME
}
```

GPAchievementType clases

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

GPAchievementState clases

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

GooglePlayResponceCode clases

```
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 befor you can accsess 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 dictoinary

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 GooglePlayResponceCode 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;
```

Game Gifting in Android

To make gameplay more collaborative and improve social engagement, your game can allow players to send and request gifts of in-game resources or items by using the game gifts API. Your game can display a built-in user interface (UI) provided by Play Games services that makes it easy for players to send and request gifts for in-game items and resources to friends in their Google+ circles. Request recipients receive notifications on all devices on which the recipients are logged in (unless notifications is disabled).

Note: The game gifts API is currently supported for Android through the Google Play services SDK.

Warning: Your game must not send, request, or accept an in-game gift without an explicit approval from a user. Doing so violates the terms of service.

The game gifts API in Play Games services is also flexible enough that your game can use it to allow players to negotiate and trade for items with each other.

There are two types of requests that players can send using the game gifts feature in Play Games services:

- A wish request to ask for in-game items or some other tangible form of assistance from their friends.
- A gift request to send in-game items or some other tangible form of assistance from their friends; for example, players can gift "lives" to each other to extend gameplay.

A player can specify one or more target request recipients from the default request sending UI. A gift or wish can be consumed(that is, accepted by a recipient) or dismissed by a recipient. Each request can be consumed only once. Requests expire after a period of time if they are not consumed.

Sending a request

To send a gift or wish request, follow these steps:

1. Call SendGiftRequest to bring up the default request sending UI so that the player can select a recipient for the request. In the call, you must specify the request type (TYPE_GIFT or TYPE_WISH). You can use the payload input parameter to provide additional data in byte array format to indicate what game-specific items are being requested or sent. The payload can be empty, as shown in the snippet below

```
GooglePlayManager.instance.SendGiftRequest(type,lifetime, icon, description, playload);
```

After the user selects one or more request recipients from the UI, the request is sent by the game gifts API to the Play Games services. Play Games services then creates a GPGameRequest object to represent the request and routes this request object to the target recipient. To learn how your game can accept an incoming request, see Handling requests.

In your game, you can subscribe to SEND_GIFT_RESULT_RECEIVED event to check if any errors occurred during the sending of the request. Event data will contain GooglePlayGiftRequestResult

```
GooglePlayManager.instance.addEventListener(GooglePlayManager.SEND_GIFT_RESULT_RECEIVED,
OnSendGifttResult);
```

```
private void OnSendGifttResult(CEvent e) {
        GooglePlayGiftRequestResult result = e.data as GooglePlayGiftRequestResult;
        Debug.Log("Send Gift Rsult: " + result.code);

        SA_StatusBar.text = "Gift Send Result:  " + result.code.ToString();
}
```

Launching your game from a request notification

By default, request recipients will see a notification appear on their devices when they receive a request from friends in their Google+ circles. If the recipient already has your

game installed on the device, the recipient can simply expand and click on the notification to launch a UI to select whether to accept the request. After the recipient makes this selection, the system automatically launches your game.

Note: If the recipient does not have your game installed on the device, the recipient is directed to the Google Play store to download the game.

Next, your game should retrieve the list of requests that the recipient selected. The list of requests are represented as GPGameRequest objects stored in the GooglePlayManager.

```
List<GPGameRequest> gameRequests = GooglePlayManager.instance.gameRequests
```

Your game should programmatically accept requests that are returned in the gameRequests. To learn how to automatically accept requests, see step 2 in Handling a request programatically.

Notifying about pending request

You can get notification when there is new pending request available. Notification will be fired if user switched to you application by clicking on pending request from PlayGames Application.

```
GooglePlayManager.instance.addEventListener
(GooglePlayManager.PENDING_GAME_REQUESTS_DETECTED, OnPendingGiftsDetected);


 private void OnPendingGiftsDetected(CEvent e) {
        AndroidDialog dialog = AndroidDialog.Create("Pending Gifts Detected", "You got
 few gifts from your friends, do you whant to take a look?");
        dialog.addEventListener(BaseEvent.COMPLETE, OnPromtGiftDialogClose);
    }


    private void OnPromtGiftDialogClose(CEvent e) {
        //removing listner
        (e.dispatcher as AndroidDialog).removeEventListener(BaseEvent.COMPLETE,
 OnPromtGiftDialogClose);


        //parsing result
```

```
        switch((AndroidDialogResult)e.data) {
        case AndroidDialogResult.YES:
            GooglePlayManager.instance.ShowRequestsAccepDialog();
            break;



        }
    }
```

Handling requests using the default inbox UI

Your game can bring up the default inbox UI provided by the SDK to let request recipients view a list of all the inbound requests that are waiting to be accepted. Recipients can then select whether to accept a request. To display the inbox UI, call ShowRequestsAccepDialog.

```
GooglePlayManager.instance.ShowRequestsAccepDialog();
```

If the call is successful, the game displays the default inbox UI and prompts recipients to select the inbound requests that they want to accept. The recipient's selection is then returned as array in event. with you can retrieve by subscribing on GAME_REQUESTS_ACCEPTED event.

```
GooglePlayManager.instance.addEventListener (GooglePlayManager.GAME_REQUESTS_ACCEPTED,
OnGameRequestAccepted);


private void OnGameRequestAccepted(CEvent e) {
        List<GPGameRequest> gifts = e.data as List<GPGameRequest>;
        foreach(GPGameRequest g in gifts) {
            AndroidNative.showMessage("Gfit Accepted", g.playload + " is excepted");
        }
    }
```

Handling requests programmatically

Your game can also accept requests programmatically. You might use this approach, for example, to display a custom request UI in your game after the user signs in. If your game

uses a custom UI, make sure that it presents users with an option to accept the request and a separate option to dismiss the request.

To accept/reject requests programmatically, use AcceptRequests / DismissRequest of GooglePlayManager

Classes Documentation

GooglePlayGiftRequestResult

Getters:

```
Game Activity Code
public GP_GamesActivityResultCodes code
```

```
True is result is success
public bool isSuccess
```

```
True is result is fai
public bool isFailure
```

GPGameRequest

Getters:

Retrieves the ID of this request.

```
public string id;
```

Retrieves the data associated with the request.

```
public string playload;
```

The server timestamp (in milliseconds from epoch) at which this request will expire.

```
public long expirationTimestamp;
```

The server timestamp (in milliseconds from epoch) at which this request was created.

```
public long creationTimestamp;
```

Retrieves the player id of request sender.

```
public string sender;
```

Retrieves the type of this request.

```
public GPGameRequestType type;
```

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:

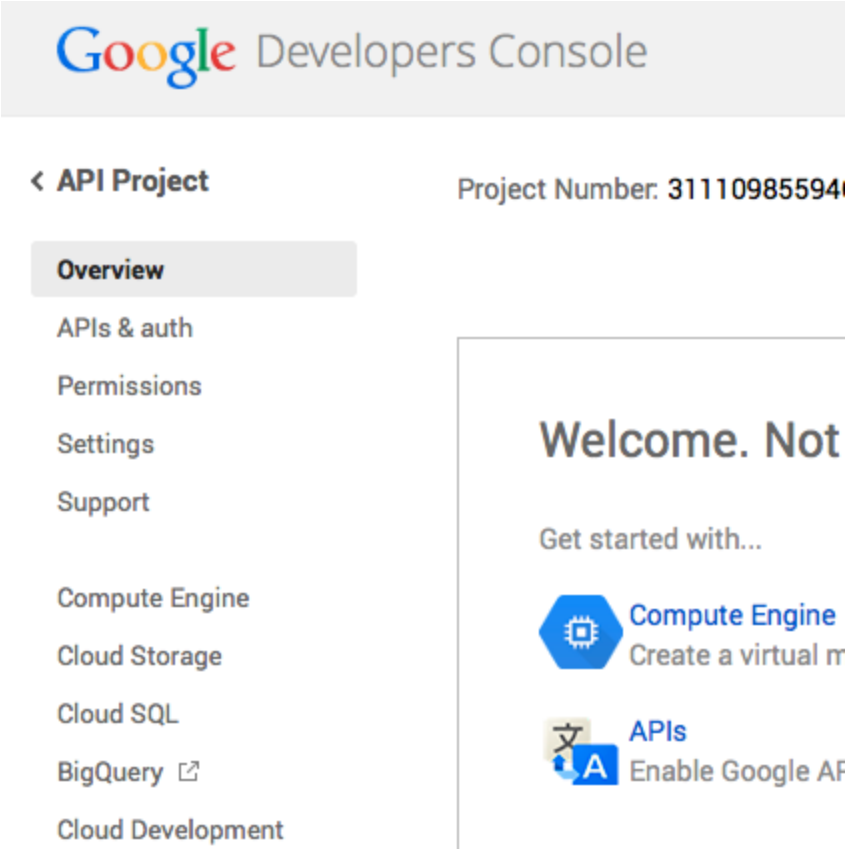
1. In the sidebar on the left, select **APIs & auth > Registered apps**.
2. Click **Register app**.
3. In the **Name** field, type your app's name.
4. Click **Android > Accessing APIs directly from Android**.
5. Under **Android identification**, type the package name for your app.
6. Enter an SHA1 fingerprint. 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

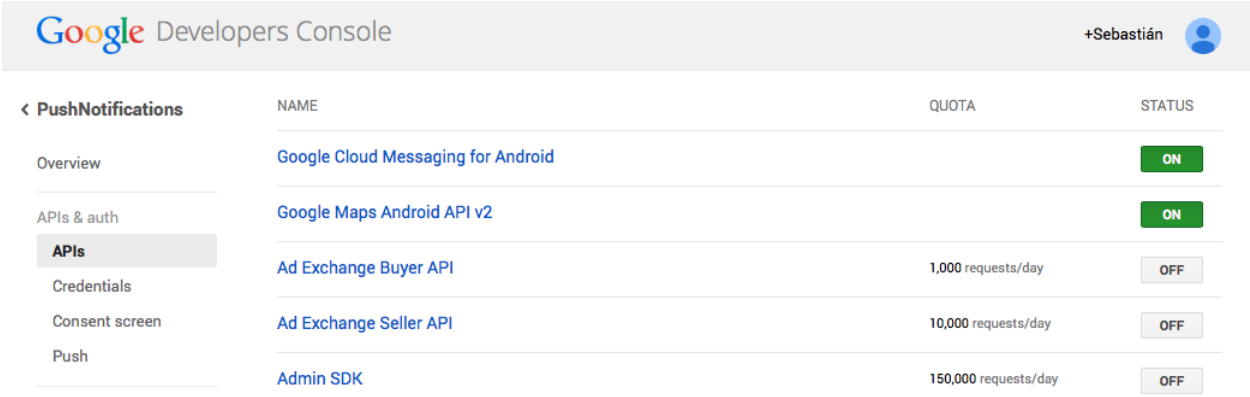
2. If you have never created a project before, the page will prompt you to create one

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

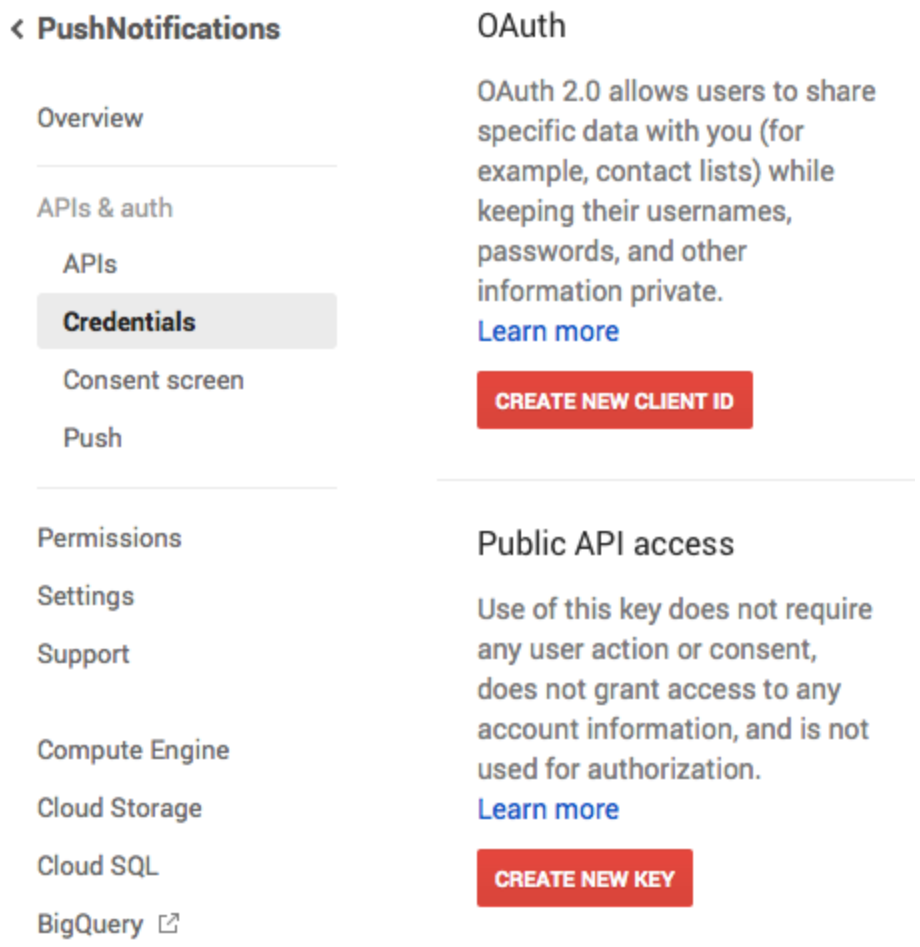


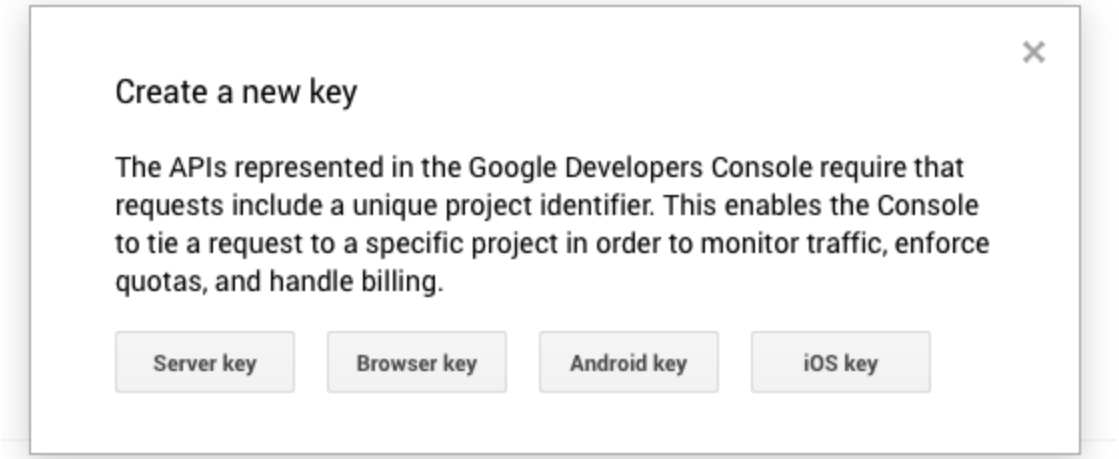
In any case, take note of the value of **Project Number** present on the screen This value will be used later when configuring the Smart Devices Main Object in GeneXus to enable it to receive notifications (*Sender ID* property).

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

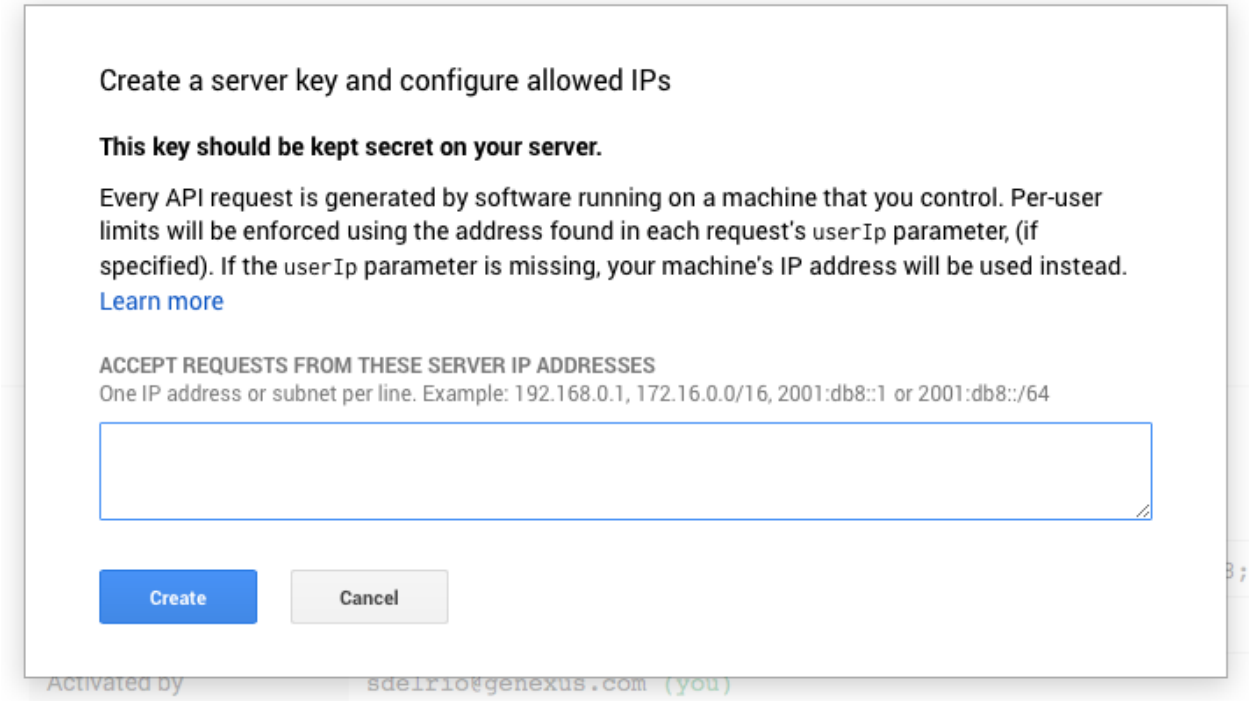


4. Generate the API Key. Go to *Credentials* option in the *APIs & auth* menu, select *Create new Key* and *Server Key.*

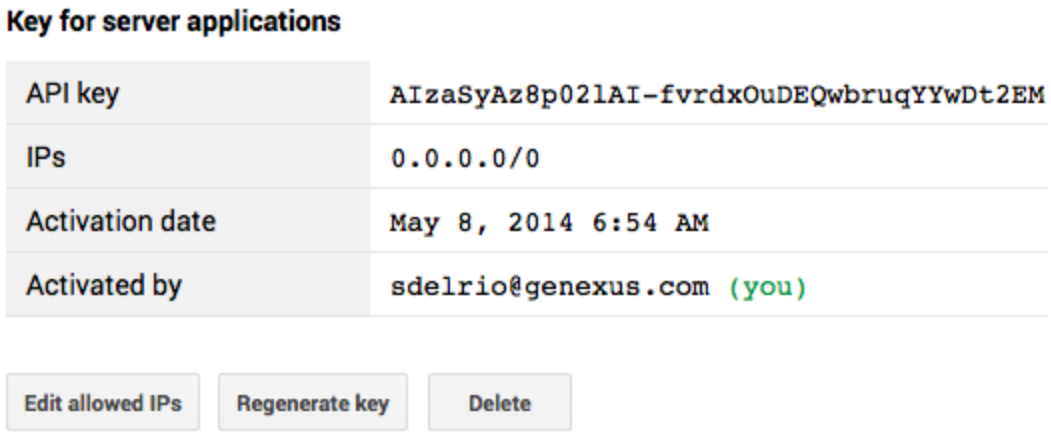




In the popup window displayed you should indicate the range of IP addresses which will have permission to send notifications to this Project ID, if you want to set "All IPs" use the value `0.0.0.0/0`



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.



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.RgisterDevice(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**(*crossplatfrom*) 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**(*crossplatfrom*) 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 a**nalytics.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. Read more about log level here.

```
public void SetlogLevel(GPLogLevel 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

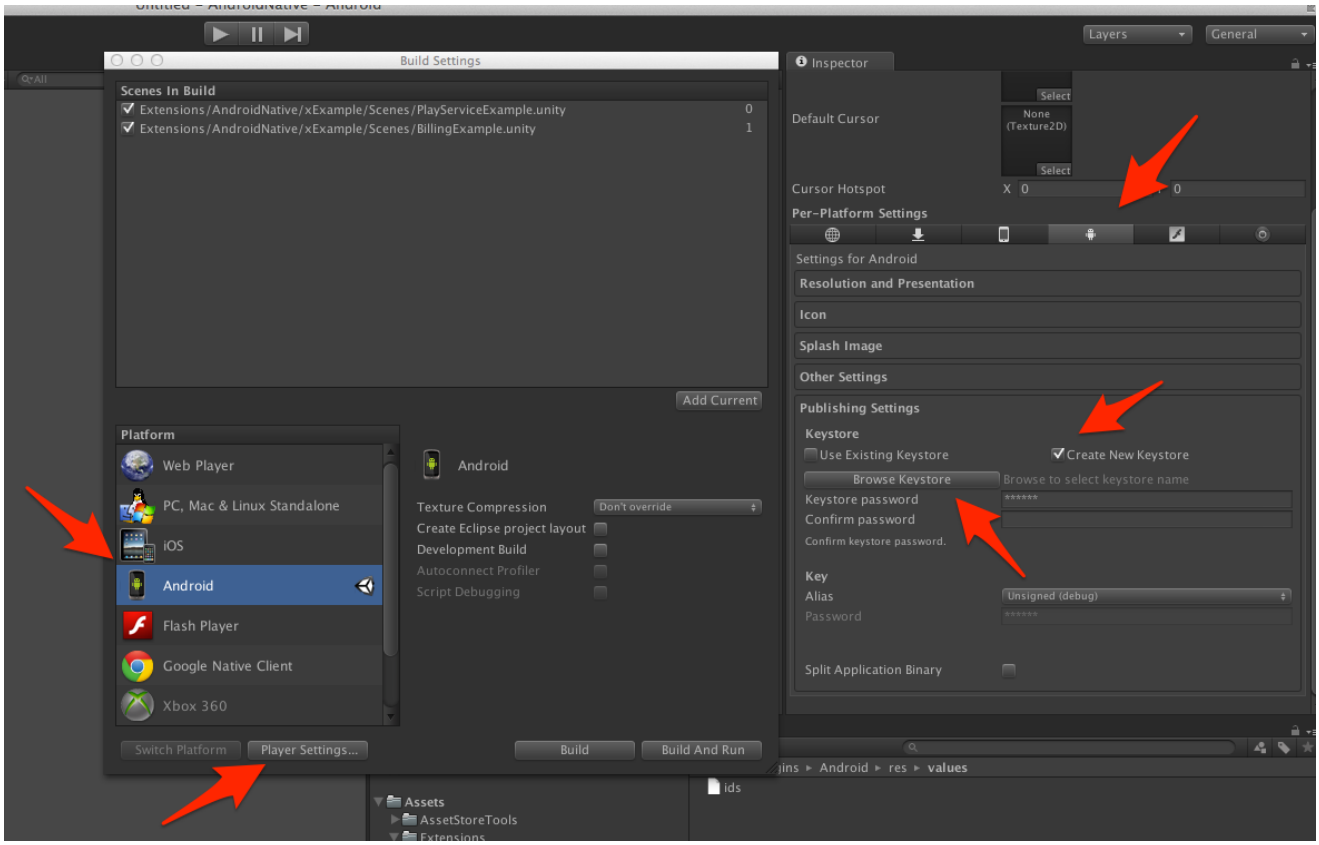
Go to:

File → Build Settings

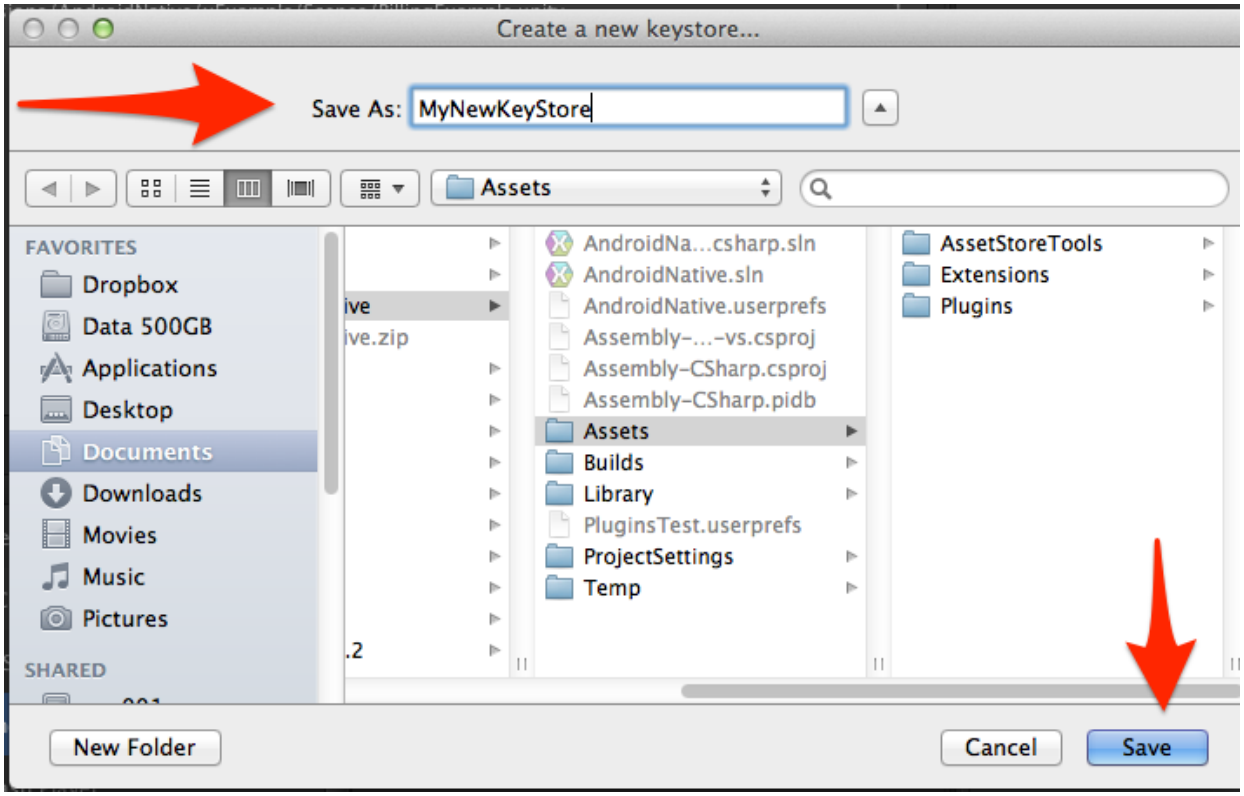
Choose Android platform and press **Player Settings** button.

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.



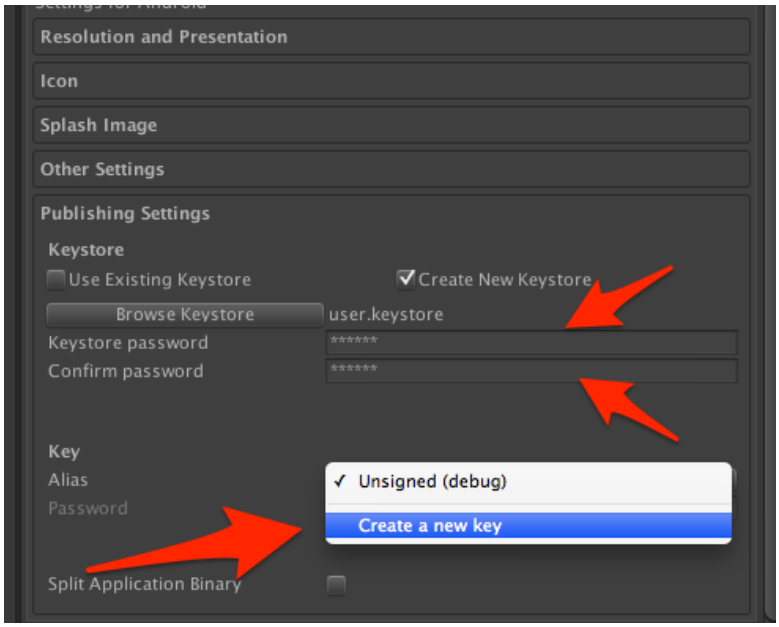
in the dialog box, select the path and name for the new keystore.



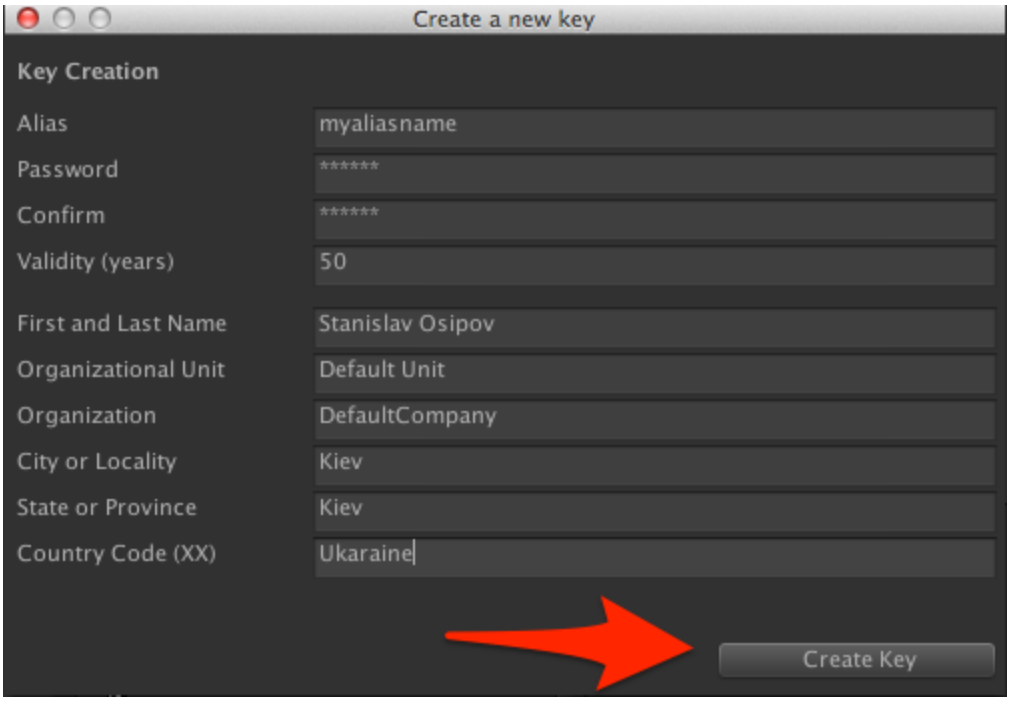
And press “**Save**” button.

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.



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_StartInterstisialAd
- AN_LoadInterstisialAd
- AN_ShowInterstisialAd

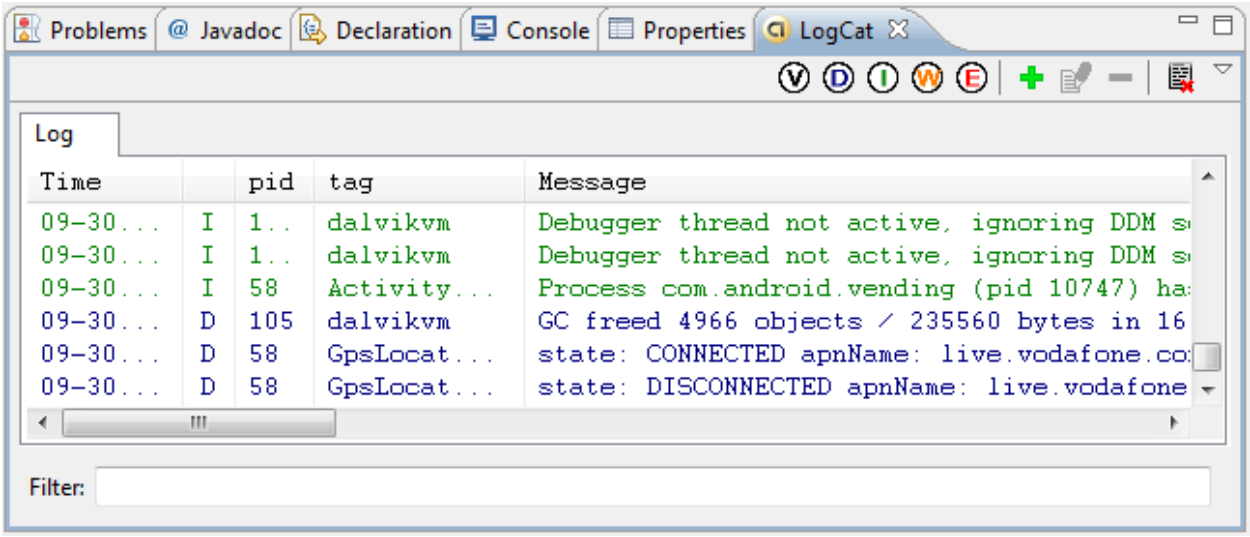
Frequently Asked Questions

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.

How to merge manifest with another android plugin?

Android Native plugin should be main activity. so this is very important lines that you should have in your new merget manifest

```
<activity android:name="com.androidnative.AndroidNativeBridge"
android:label="@string/app_name"
android:configChanges="fontScale|keyboard|keyboardHidden|locale|mnc|mcc|navigation|orientation|screenLayout|screenSize|smallestScreenSize|uiMode|touchscreen" android:launchMode="singleTask" android:screenOrientation="landscape">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
. . . . . .
</activity>
```

Some of plugin feature also may have dependencies from manifest. That's why AndroidManifest you got with the plugin contains markers in manifest like this:

```
 <!-- Google Mobile Ad Block Start -->
    <activity android:name="com.google.android.gms.ads.AdActivity"
android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|uiMode|screenSize|smallestScreenSize" />
 <!-- Block End -->


<!-- Facebook Block Start -->
    <meta-data android:name="com.facebook.sdk.ApplicationId" android:value="\
```

```
395891937214418" />
    <activity
        android:name="com.facebook.LoginActivity"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" />

    <activity android:name="com.facebook.unity.FBUnityLoginActivity"
android:theme="@android:style/Theme.Translucent.NoTitleBar.Fullscreen">
    </activity>
    <activity android:name="com.facebook.unity.FBUnityDeepLinkingActivity"
android:exported="true">
    </activity>
<!-- Block End →
```

So if you want to use the Google Mobile Ad feature, you need to transfer all Google Mobile Ad blocks to your updated manifest.

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, with 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/Scripts/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

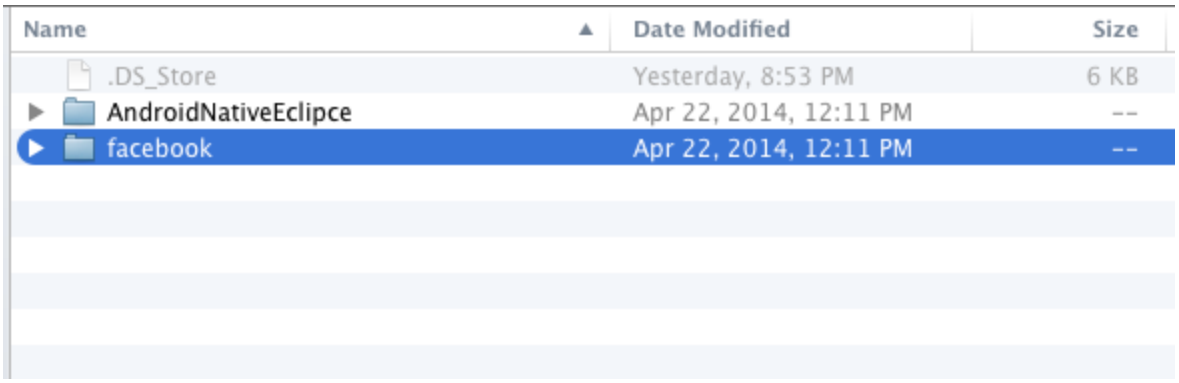
First of all we need to unrar the project.

Project archive can be found at:

Asstes/Extensions/GooglePlayCommon/Eclipse/ANEclipseProject.rar

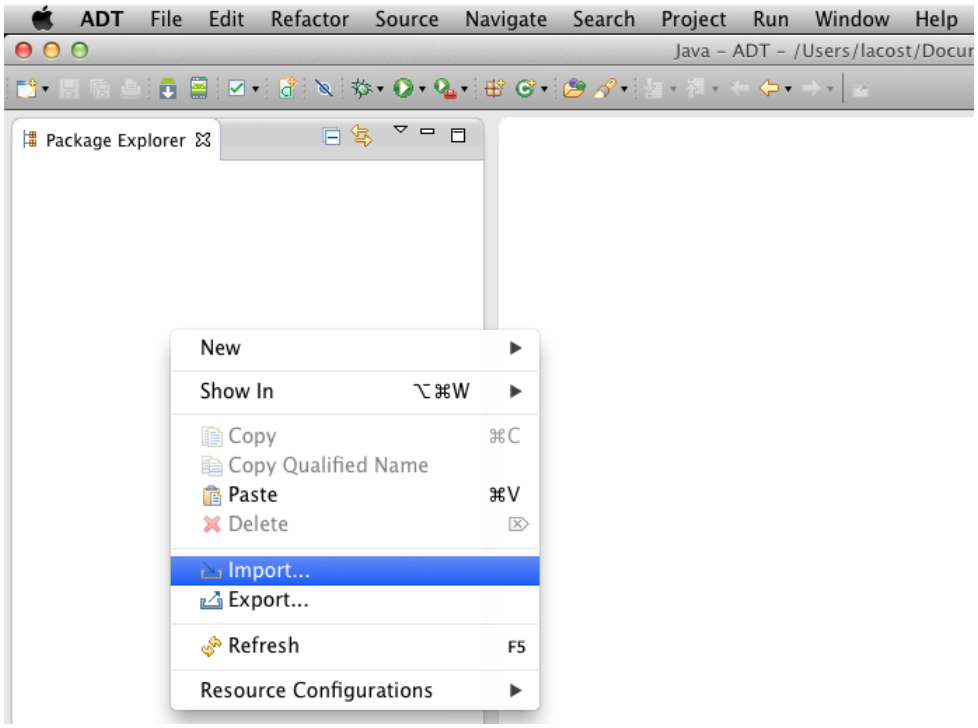
You can replace this rar file to any comfortable place for your eclipse project and unrar in there.

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

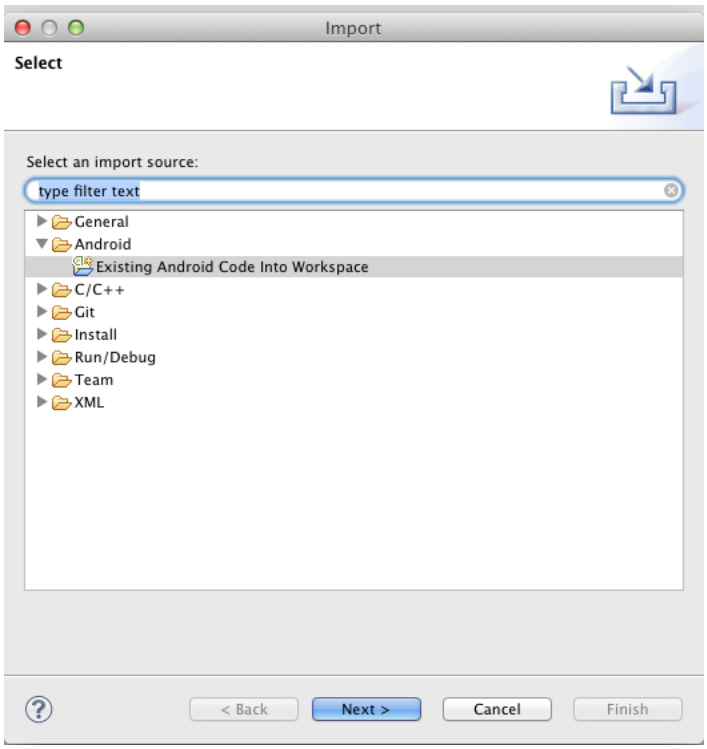


Now we need to import those project to ADT. Run the android ADT (Eclipse) you can get it from Android SDK download page.

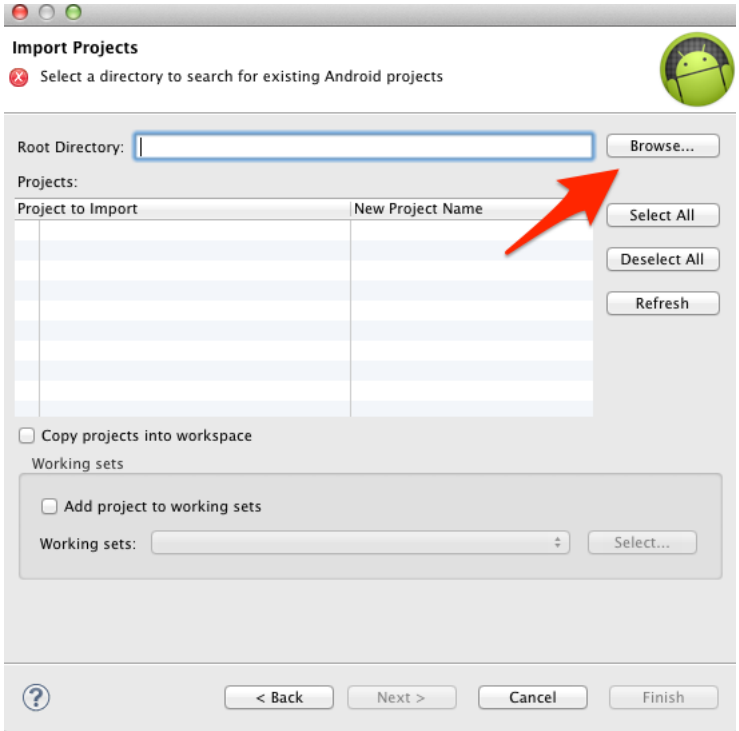
After ADT is launched right click on **Package Explorer** tab and choose **Import** menu item.



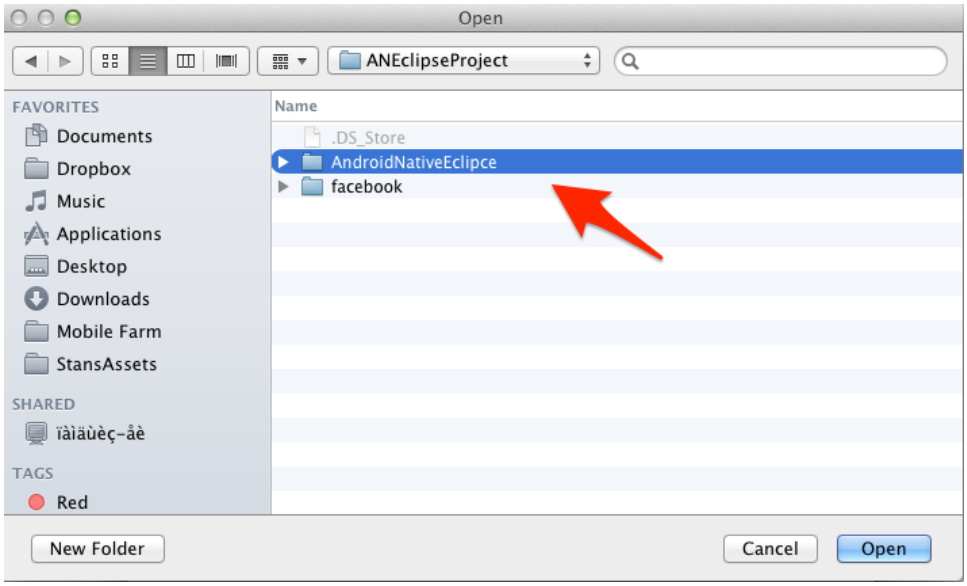
In the import setting choose **Android** → **Existing Android Code Into Workspace**



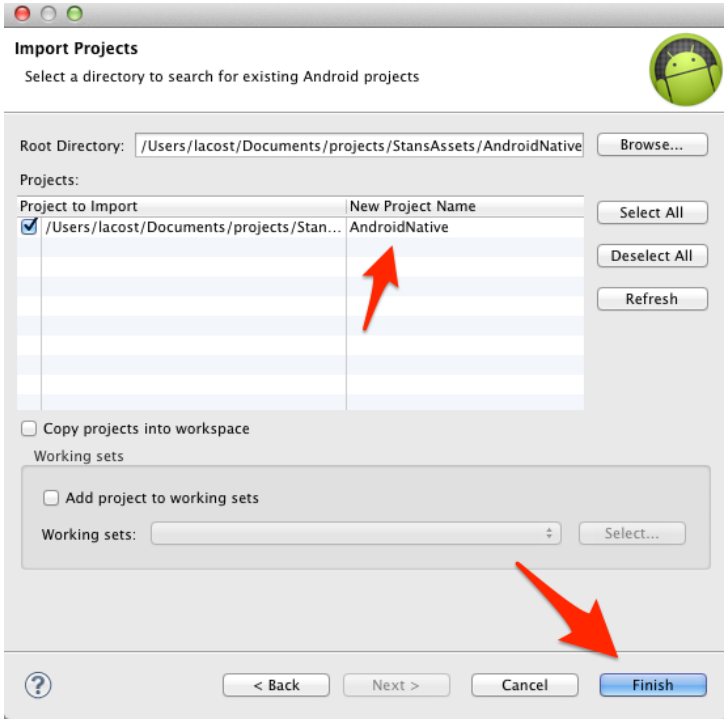
Hit next and press browse button.



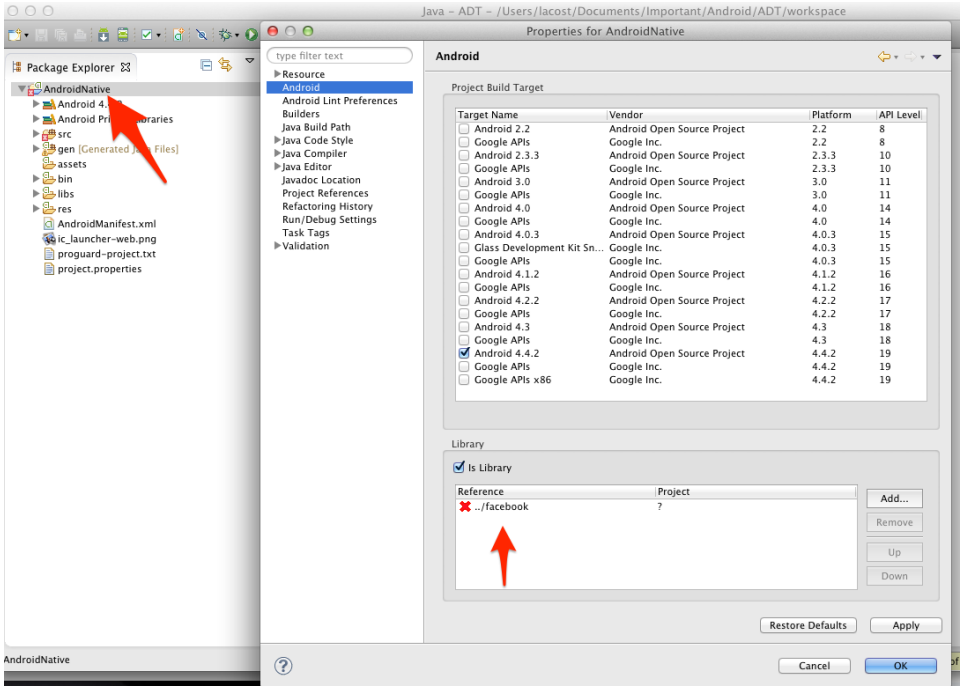
Browse to the **AndroidNativeEclipce** project location and choose **AndroidNativeEclipce** folder and press **Open**.



Then just press **Finish** button.

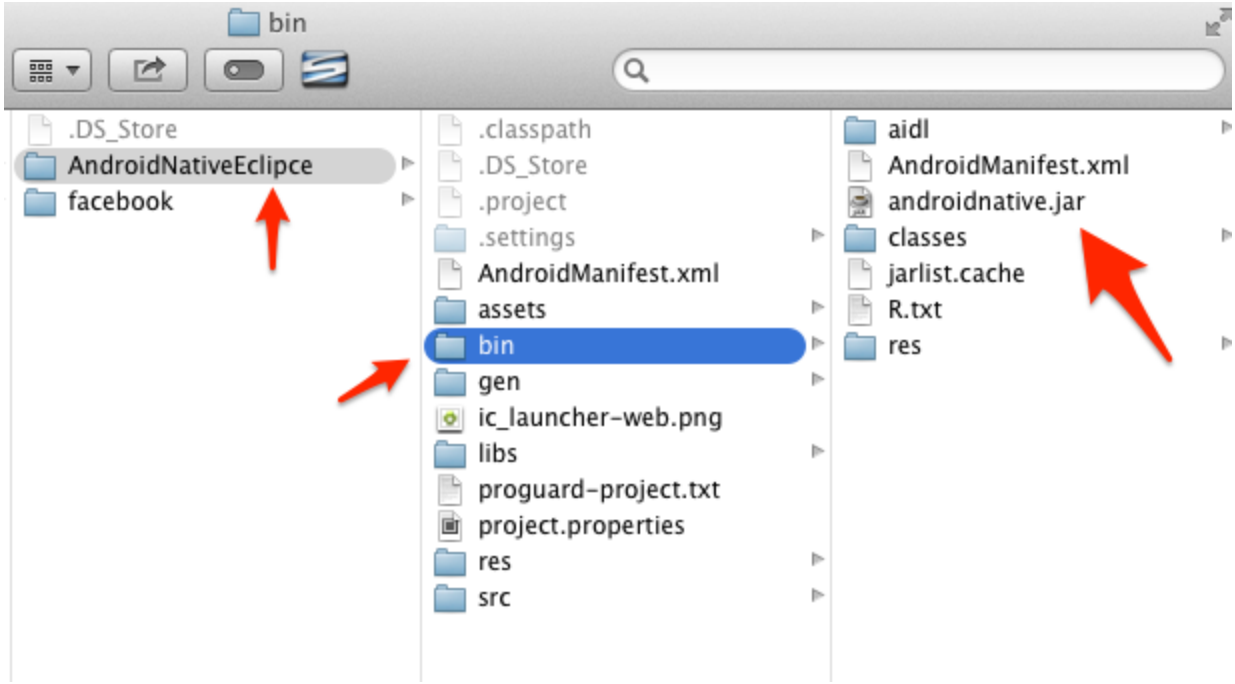


You will see **AndroidNative** project imported to the ADT. It will have compilation errors because if we will open the Project Setting tab we will see that it dependent from facebook project. To get rid of those error we should import **facebook** project in the smae way we did with **AndroidNativeEclipce**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

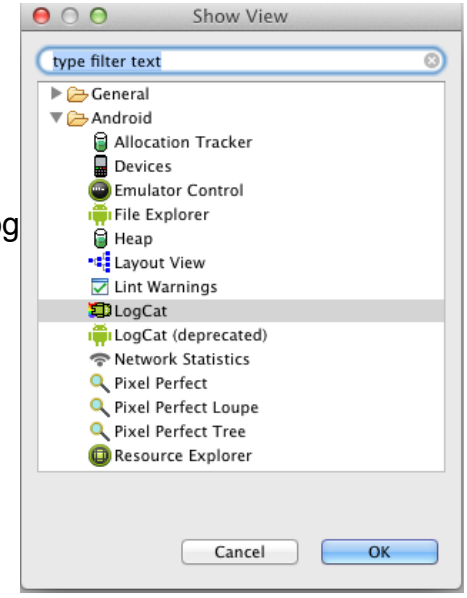


That it, now you can add your stuff and replace **androidnative.jar** in your Unity project

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** ot **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 AndroidNativeBridje.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

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

GoogleAdExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Admob. It will describe how you can use Google Mobile Ad API. The controller script Android Google Ads Example is attached to the _Controller gameobject and provides example for API calls.

*GoogleAdExample[**Setup**]*

The scene should work out of the box. But it uses my ad identification. In order to see ad from your account you need to complete your ad account set up and replace ad unit id's in the **AndroidGoogleAdsExample**

```
private const string MY_BANNERS_AD_UNIT_ID = "ca-app-pub-6101605888755494/1824764765";
private const string MY_INTERSTISIALS_AD_UNIT_ID ="ca-app-pub-6101605888755494/3301497967";
```

GoogleAdPrefabSolutionExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Admob. It will describe how you can use Google Mobile Ad API **without any actions**. You have two controller scripts AndroidAdMobBanner and AndroidAdMobBannerInterstitial are attached to the *Banner Ad* and *Interstitial Ad* gameobjects and provides examples for API calls.

*GoogleAdPrefabSolutionExample[**Setup**]*

The scene should work out of the box. This example use my identifiers. You need change param of gameobjects. For Banner - **BannersUnityId**(your_ad_unit_ad), **Size** of your banner, **Anchor**, **Width**, **Height**, and for Interstitial - **InterstitialUnityId**.

BillingExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Billing. It will describe how you can use Billing inAppPurchases. The controller script BillingExample is attached to the _Controller gameobject and provides example for API calls.

*BillingExample[**Setup**]*

You must finish start setup. In order to start purchase you need add test account at Google Developer Console and complete your billing setUp for testPurchase. Replace SKU(identifier) at GPaymentManagerExample **ANDROID_TEST_PURCHASED:**

```
public const string ANDROID_TEST_PURCHASED = "android.test.purchased";
```

BillingImplementationExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Billing. It will describe how you can use Billing inAppPurchases. The controller script GamePlayExample is attached to the _Controller gameobject and provides example for API calls.

*BillingImplementationExample[**Setup**]*

You must finish start setup. In order to start purchase you need add test account at Google Developer Console and complete your billing setUp for testPurchase. After connection you can try AddCoins or AddBoost by clicking on the appropriate button.

AddCoins - consumable product. AddBoost - consumable product. Don`t forget change bundle_id and sdk version at AndroidManifest.xml.

NativePopUpsEx

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Other. It will describe how you can use Native PopUps. The controller script AndroidPopUpExamples is attached to the _Controller gameobject and provides example for API calls.

*NativePopUpsEx[**Setup**]*

You need entered your rate text and link for rate your app - **rateUrl**. You can take **Android Rate Pop Up**, **Android Dialog Pop Up**, **Android Message Pop Up** and **Preloader** for your app:

```
private void ShowPreloader() {
    Invoke("HidePreloader", 2f);
    AndroidNativeUtility.ShowPreloader("Loading", "Wait 2 seconds please");
}
```

Notifications

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Other. It will describe how you can use Native Notifications API. The controller script NotificationsExample is attached to the _Controller gameobject and provides example for API calls.

*Notifications[**Setup**]*

In order to see simple notifications you can use this methods:

```
private void Toast() {
    AndroidToast.ShowToastNotification ("Hello Toast", AndroidToast.LENGTH_LONG);
}

private void Local() {
    AndroidNotificationManager.ScheduleLocalNotification("Hello", "This is notify", 5);
}
```

For using google cloud messaging you must complete this setup and you need replace SENDER_ID in the **NotificationsExample.**

```
public const string SENDER_ID = "YOUR_SENDER_ID_HERE";
private void Reg() {
    GoogleCloudMessageService.instance.RgisterDevice(SENDER_ID);
}
private void LoadLastMessage() {
    GoogleCloudMessageService.instance.LoadLastMessage();
}
```

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.RgisterDevice(SENDER_ID);

OtherFeatures

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Other. It will describe how you can use features of emmersive mode. The controller script AnOtherFeaturesPreview is attached to the _Controller gameobject and provides example for API calls.

OtherFeatures[Setup]

Activate emmersive mode at example scene.

PlayServiceExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/PlayService. It will describe how you can use google play services API. The controller script PlayServiceExample is attached to the _Controller gameobject and provides example for API calls.

*PlayServiceExample[**Setup**]*

You will have to complete all the installation instructions achievements and leaderboards here. After that copy your app_name, names and ids of leaderboards and achievements to **ids.xml** which has path Assets/Plugin/Android/res/values/. Enter your bundle_id, version code and version of project.

Enter to PlayServiceExample:

```
private const string LEADERBOARD_NAME = "REPLACE_WITH_YOUR_NAME";

private const string LEADERBOARD_ID = "CgkIipfs2qcGEAIQAA";
```

GooleCloudExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/PlayService. It will describe how you can use google cloud API. The controller script GoogleCloudUseExample is attached to the _Controller gameobject and provides example for API calls.

*GooleCloudExample[**Setup**]*

Complete small setup. Don`t forget about this permission at manifest:

```
<manifest ...>
```

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

      ...
  </application>
</manifest>
```

CustomLeaderborUIExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/PlayService. It will describe how you can use google play services API. The controller script PlayServiceCustomLBExample is attached to the _Controller gameobject and provides example for API calls.

*CustomLeaderborUIExample[**Setup**]*

This example is in a more expanded form work leaderboard. Enter your id:

```
private const string LEADERBOARD_ID = "REPLACE_WITH_YOUR_ID";
```

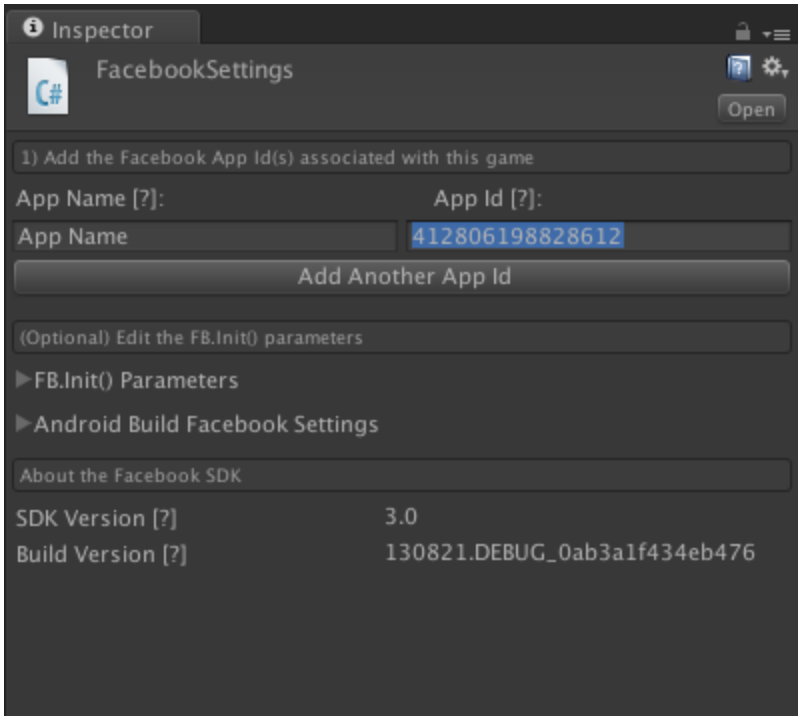
The rest of the code is similar to the use in the scene playServiceExample.

FacebookExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Social. It will describe how you can use facebook API. The controller script FacebookAndroidUseExample is attached to the _Controller gameobject and provides example for API calls.

*FacebookExample[**Setup**]*

You need complete facebook setup. Copy **app_name** and **app_id** from FB developers and enter to Facebook Menu -> Edit settings:



Two ways of using key hash:

If you chose the way of getting the connection to facebook via the debug key hash, you should FB and in Unity to specify the debug key hash. If simple key, then do the same manipulations

---

TwitterExample

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Social. It will describe how you can use twitter API. The controller script TwitterAndroidUseExample is attached to the _Controller gameobject and provides example for API calls.

*TwitterExample[**Setup**]*

You need complete twitter app setup. Enter to TwitterAndroidUseExample your app_key and app_secret_key:

```
private static string TWITTER_CONSUMER_KEY = "YOUR_KEY";
private static string TWITTER_CONSUMER_SECRET = "YOUR_SECRET_KEY";
```

SocialSharing

This example scene can be found at Assets/Extensions/AndroidNative/xExample/Scenes/Social. It will describe how you can use native sharing API. The controller script AndroidSocialNativeExample is attached to the _Controller gameobject and provides example for API calls.

*SocialSharing[**Setup**]*

Posting with a choice of resource:

```
public void ShareScreehshot() {
    StartCoroutine(PostScreenshot());
}
public void ShareImage() {
    AndroidSocialGate.StartShareIntent("Hello Share Intent", "Sharing Hello wolrd
image", shareTexture);
}
```

Posting without a choice of resource:

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

```
}


public void ShareMail() {
    AndroidSocialGate.StartShareIntentWithSubject("Hello Share Intent", "This is my
text to share", "My E-mail Subject", shareTexture,  "mail");
}


public void InstaShare() {
    AndroidSocialGate.StartShareIntent("Hello Share Intent", "This is my text to
share", shareTexture, "insta");
}

public void GoogleShare() {
    AndroidSocialGate.StartShareIntent("Hello Share Intent", "This is my text to
share", shareTexture, "com.google.android.apps.plus");
}
```