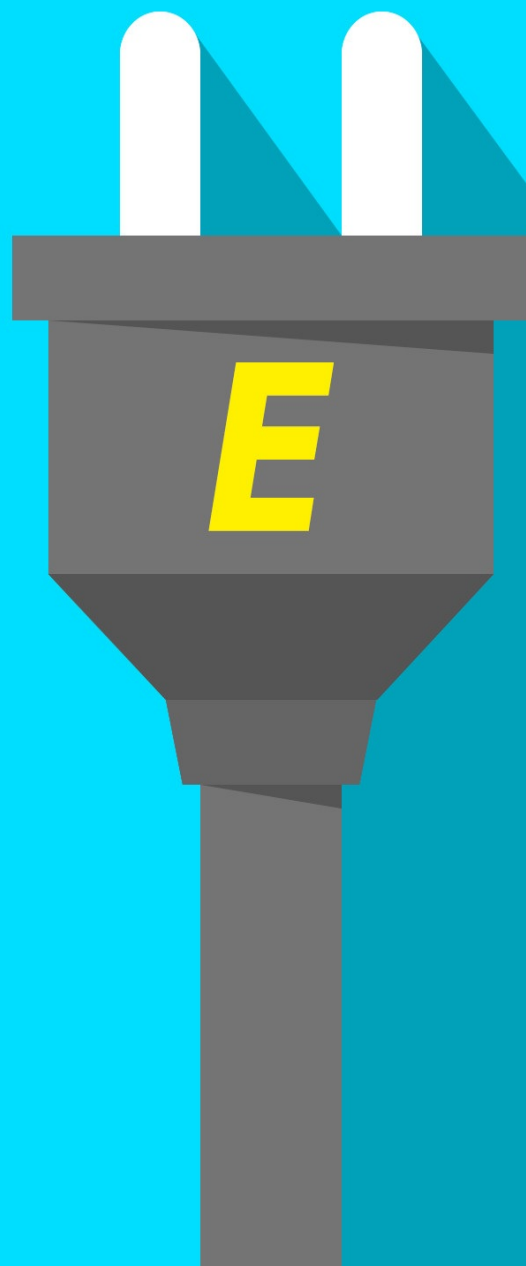


# **EASY MOBILE**

## **User Guide**



---

# Table of Contents

|                                   |           |
|-----------------------------------|-----------|
| Getting Started                   | 1.1       |
| Introduction                      | 1.1.1     |
| Requirements                      | 1.1.2     |
| Using Easy Mobile                 | 1.1.3     |
| Advertising                       | 1.2       |
| Module Configuration              | 1.2.1     |
| Setup Ad Networks                 | 1.2.1.1   |
| AdMob                             | 1.2.1.1.1 |
| Chartboost                        | 1.2.1.1.2 |
| Heyzap                            | 1.2.1.1.3 |
| Unity Ads                         | 1.2.1.1.4 |
| Automatic Ad Loading              | 1.2.1.2   |
| Default Ad Networks               | 1.2.1.3   |
| Scripting                         | 1.2.2     |
| In-App Purchasing                 | 1.3       |
| Module Configuration              | 1.3.1     |
| Enable Unity IAP                  | 1.3.1.1   |
| Target Android Store              | 1.3.1.2   |
| Receipt Validation                | 1.3.1.3   |
| Product Management                | 1.3.1.4   |
| IAP Constants Generation          | 1.3.1.5   |
| Scripting                         | 1.3.2     |
| Game Service                      | 1.4       |
| Module Configuration              | 1.4.1     |
| Android-Specific Setup            | 1.4.1.1   |
| Auto Initialization               | 1.4.1.2   |
| Leaderboards & Achievements       | 1.4.1.3   |
| Game Service Constants Generation | 1.4.1.4   |
| Scripting                         | 1.4.2     |
| Notification                      | 1.5       |

---

---

|                      |       |
|----------------------|-------|
| Module Configuration | 1.5.1 |
| Scripting            | 1.5.2 |
| Native Sharing       | 1.6   |
| Scripting            | 1.6.1 |
| Native UI            | 1.7   |
| Scripting            | 1.7.1 |
| Release Notes        | 1.8   |

---

# Easy Mobile User Guide

This document is the official user guide for Easy Mobile, a Unity plugin by SgLib Games.

## Important Links

- [Easy Mobile on Unity Asset Store](#)
- [Online Documentation](#)
- [Demo APK](#)
- [Lite Version](#)

## Connect with SgLib Games

- [Unity Asset Store](#)
- [Facebook](#)
- [Twitter](#)
- [YouTube](#)

# Introduction

Easy Mobile is our attempt to create a many-in-one Unity package that greatly simplifies the implementation of de facto standard features of mobile games including advertising, in-app purchasing, game service, notification and native mobile functionality. It does so by providing a friendly editor for setting up and managing things, and a cross-platform API which allows you to accomplish most tasks with only one line of code. It also leverages official plugins wherever possible, e.g. [Google Play Games plugin for Unity](#), to ensure reliability and compatibility without reinventing the wheel.

Easy Mobile supports two major mobile platforms: iOS and Android.

This plugin is currently divided into 6 modules, each can be enabled or disabled as needed.

- **Advertising**

- AdMob, Chartboost, Heyzap (with ad mediation) and Unity Ads
- Automatic ad loading
- Allows using multiple ad networks in one game
- Allows different ad configurations for different platforms

- **In-App Purchasing**

- Leverages Unity In-App Purchasing service
- Custom editor for easy management of product catalog
- Receipt validation

- **Game Service**

- Leverages Unity's GameCenterPlatform on iOS and Google Play Games plugin on Android
- Custom editor for easy management of leaderboards and achievements

- **Notification**

- Compatible with [OneSignal](#), a free and popular service for push notifications

- **Native Sharing**

- Shares texts and images to social networks using the native sharing functionality

- **Native UI**

- Alerts and dialogs
- Toasts (Android only)

The modular approach helps avoid inflating the build size by allowing you to enable only the modules that you use.

# Requirements

- Unity 5.3.0 or above.
- [CocoaPods](#) if using Google Mobile Ads (AdMob) on iOS.

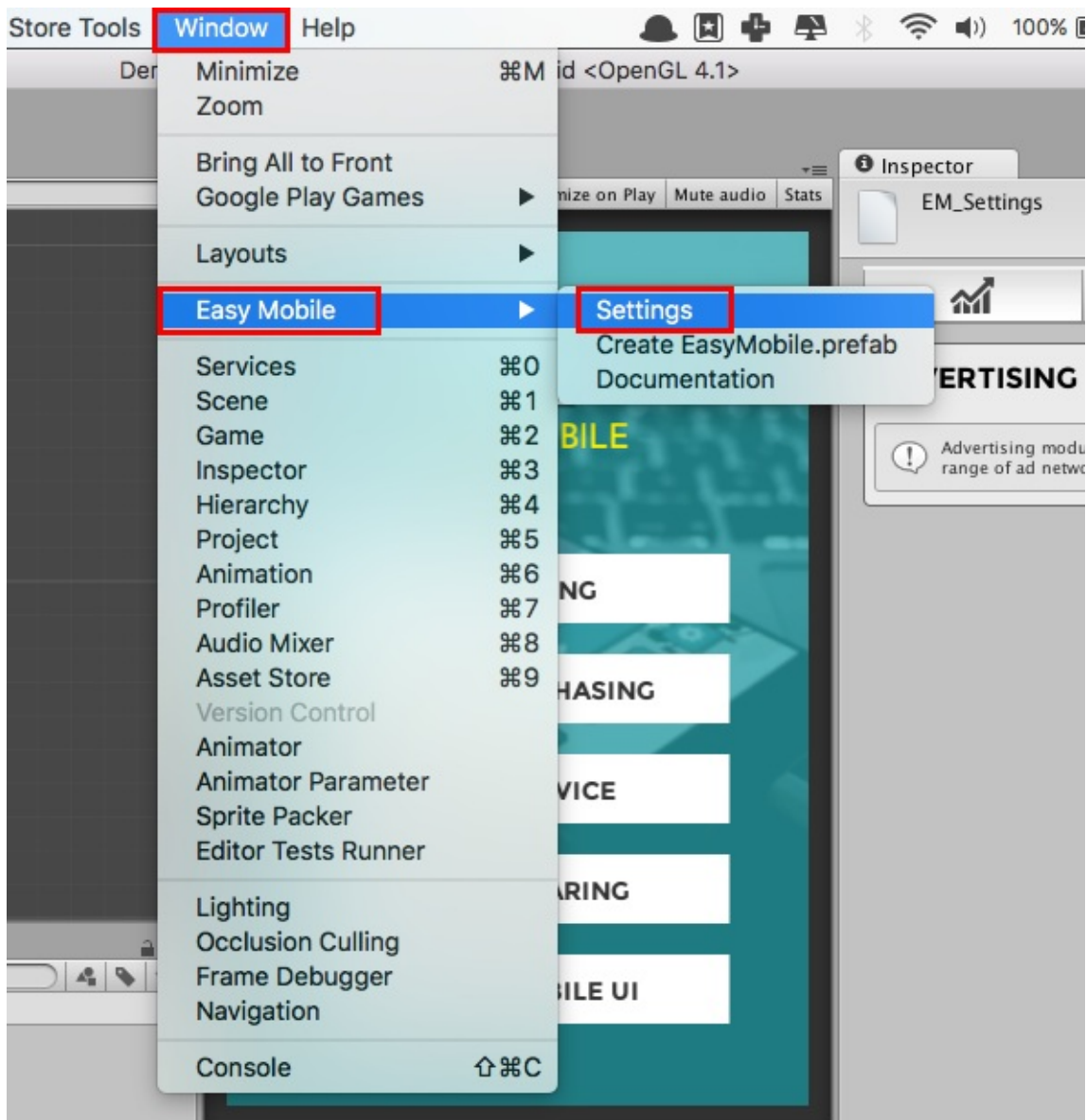
# Using Easy Mobile

Using Easy Mobile involves 3 tasks:

- Configure the plugin using the built-in Settings interface
- Make sure an instance of the EasyMobile prefab is added to your first scene
- Make appropriate API calls from script

## Configuration

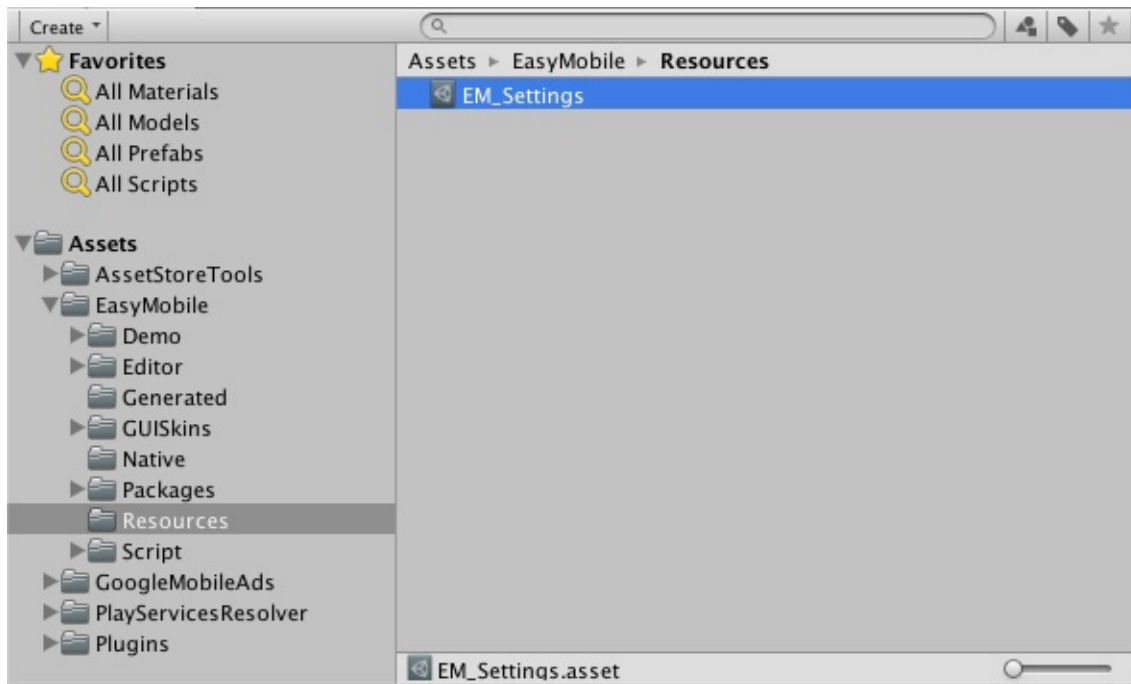
After importing Easy Mobile, there will be a new menu added at *Window > Easy Mobile* from which you can access the Settings interface and configure various modules of the plugin.





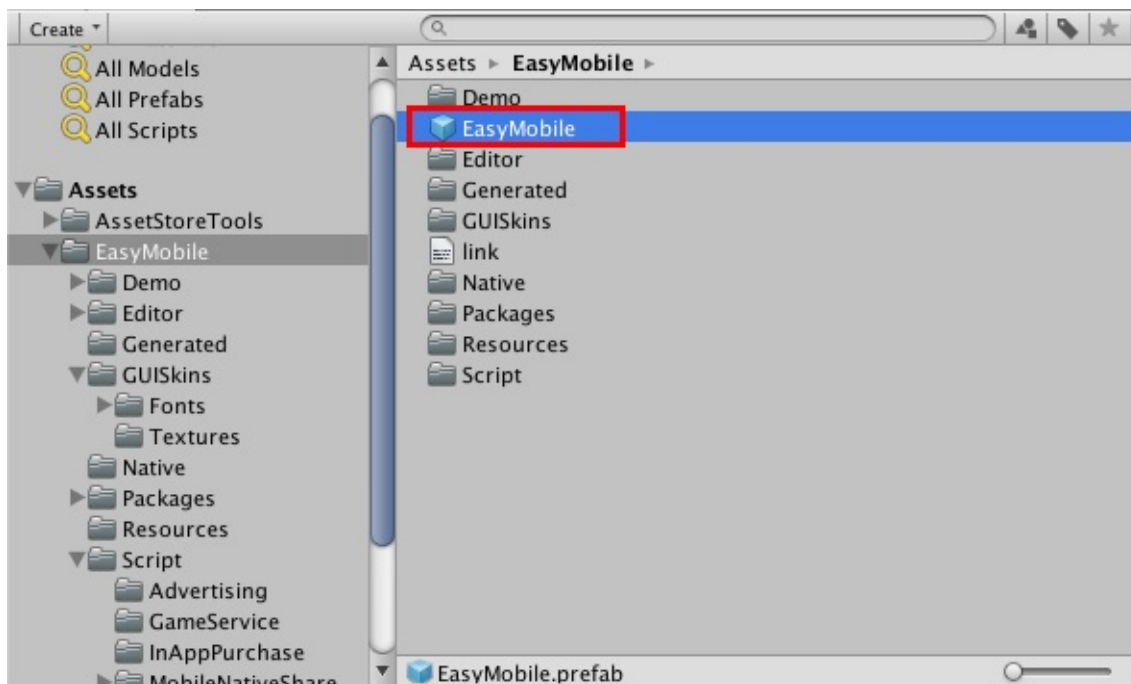
The Settings interface is the only place you go to configure the plugin. Here you can enable or disable modules, provide ads credentials, add leaderboards, create a product catalog, etc.

All these settings are stored in the EM\_Settings object, which is a ScriptableObject created automatically after importing the plugin and is located at *Assets/EasyMobile/Resources*. You can also access this EM\_Settings class from script and via its properties accessing each module settings in runtime.



## EasyMobile Prefab

For the plugin to function properly it is required that the EasyMobile prefab is added to one of the game scenes. The prefab is automatically created when importing the plugin and is located at its root folder. It will handle tasks like initialization and automatic ad loading.



It is advisable to add the EasyMobile prefab to the first scene in your game so that the modules have time to initialize before you actually use them. Likewise, this will allow the automatic ad loading process to start soon and the ads will be more likely available when needed.

## Scripting

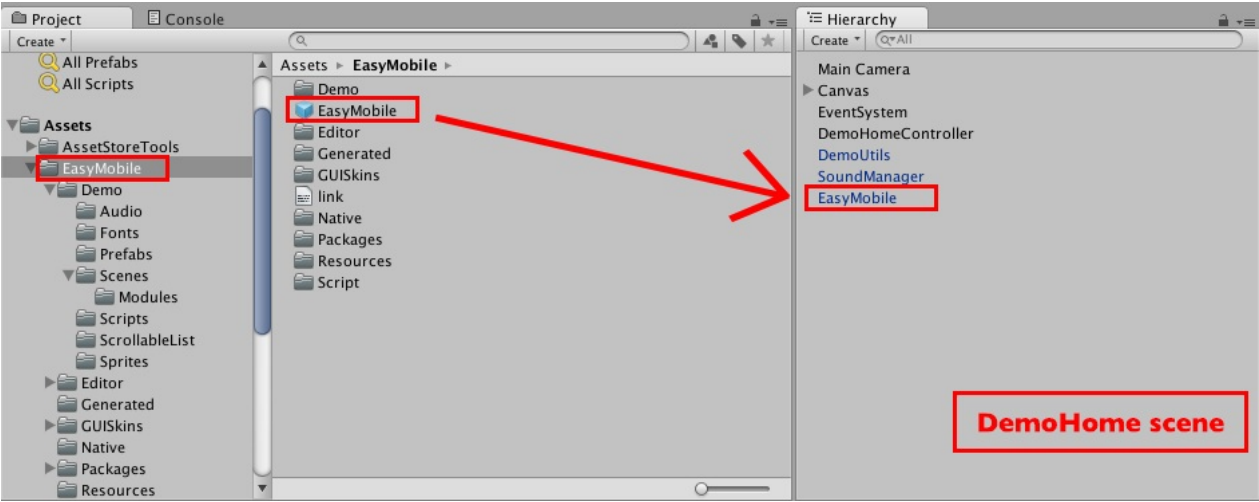
Easy Mobile API is written in C# and is put under the namespace EasyMobile. Therefore, you need to add the following statement to the top of your script in order to access its API methods.

```
using EasyMobile;
```

Easy Mobile's API is cross-platform so you can use the same codebase for both iOS and Android.

## Testing Using the Demo App

Easy Mobile comes with a demo app that you can use to quickly test each module's operation after configuring. The demo app is contained in folder *Assets/EasyMobile/Demo*. To use the demo app, you need to add the EasyMobile prefab to the *DemoHome* scene located in the *Assets/EasyMobile/Demo/Scenes* folder.



# Advertising

The Advertising module helps you quickly setup and show ads in your games. Here're some highlights of this module:

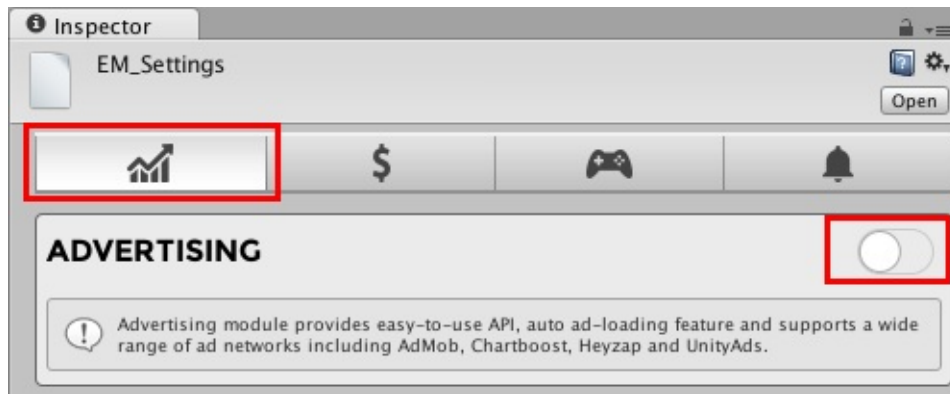
- **Supports multiple networks**
  - This module allows showing ads from most of top ad networks: AdMob, Chartboost, Heyzap and Unity Ads
  - Even more networks can be used via Heyzap mediation
- **Using multiple networks in one game**
  - It's possible to use multiple ad networks at the same time, e.g. use AdMob for banner ads, while using Chartboost for interstitial ads and Unity Ads for rewarded ads
  - Different configurations for different platforms are allowed, e.g. use Unity Ads for rewarded ads on Android, while using Chartboost for that type of ads on iOS
- **Automatic ad loading**
  - Ads will be fetched automatically in the background; new ad will be loaded if the last one was shown

The table below summarizes the ad types supported by Easy Mobile for each ad network.

| Ad Network | Banner Ad | Interstitial Ad | Rewarded Ad |
|------------|-----------|-----------------|-------------|
| AdMob      | Yes       | Yes             | No          |
| Chartboost | No        | Yes             | Yes         |
| Heyzap     | Yes       | Yes             | Yes         |
| Unity Ads  | No        | Yes             | Yes         |

# Module Configuration

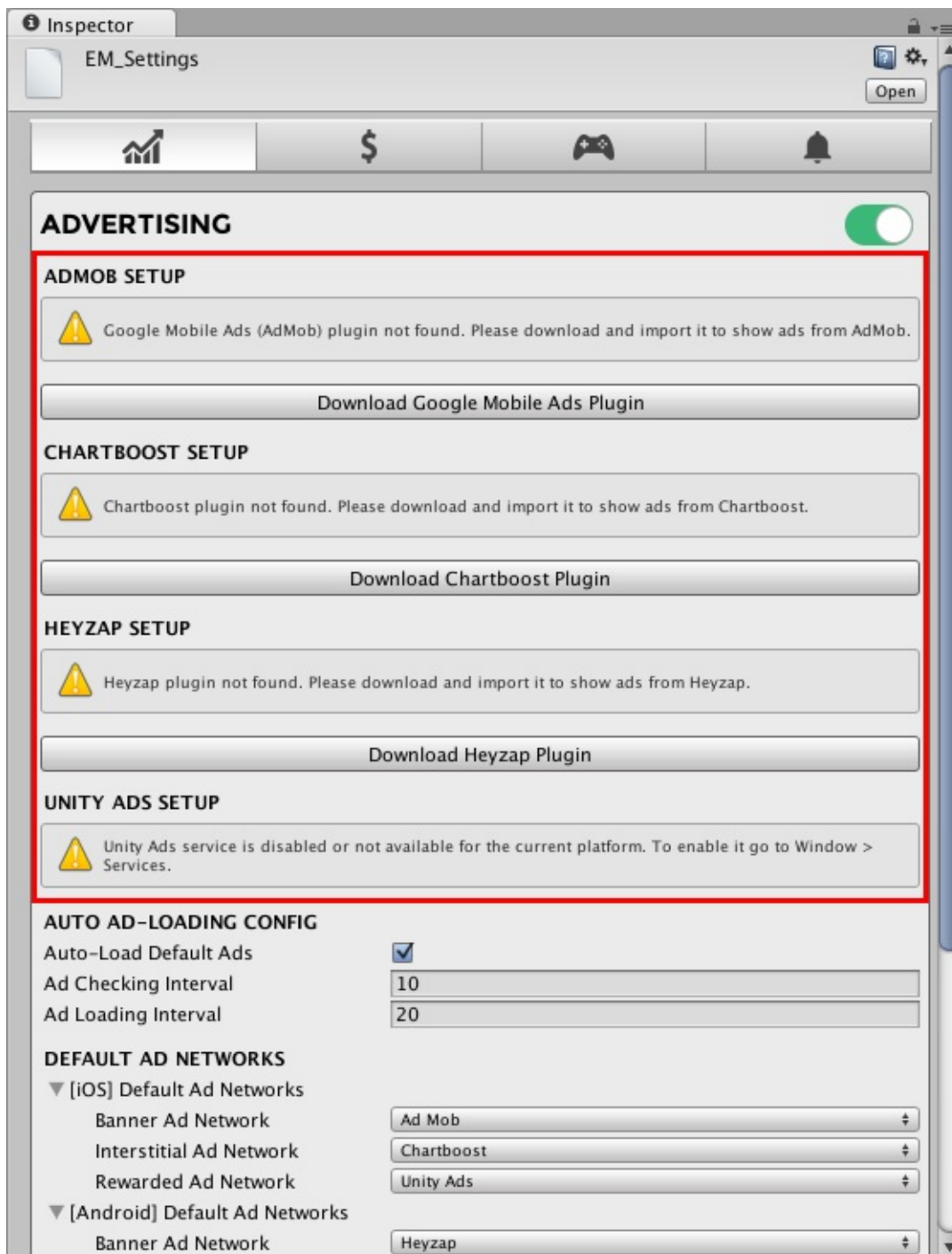
To use the Advertising module you must first enable it. Go to *Window > Easy Mobile > Settings*, select the Advertising tab, then click the right-hand side toggle to enable and start configuring the module.



# Setup Ad Networks

The Advertising module works with top mobile ad networks: AdMob, Chartboost, Heyzap and Unity Ads. To show ads from a certain network you need to import its plugin (or enable the corresponding Unity service in case of Unity Ads). Easy Mobile will automatically check for the availability of these plugins and prompt you to download and import them if needed.

Only import plugins for the ad networks you use to not increase the build size unnecessarily.

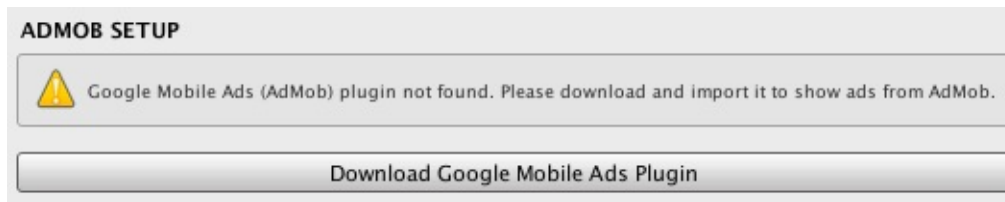




# AdMob

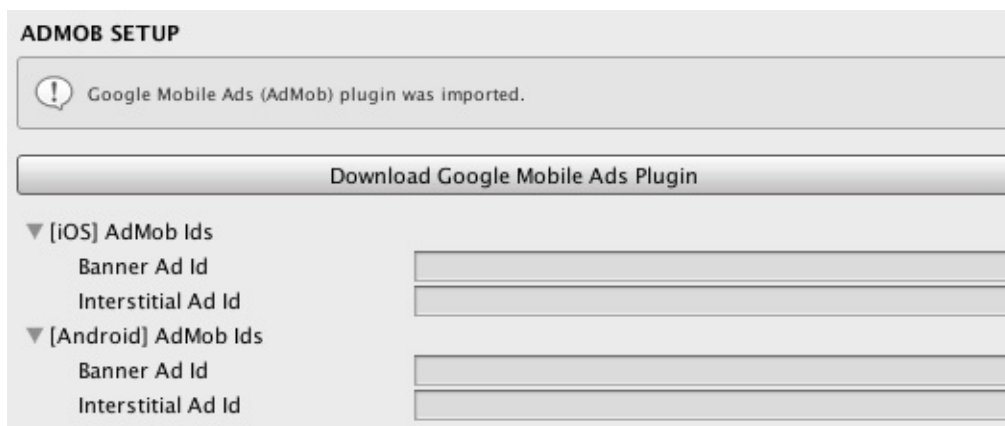
## Import AdMob Plugin

To show ads from AdMob you need to import the [Google Mobile Ads plugin](#). In the **ADMOB SETUP** section, click the *Download Google Mobile Ads Plugin* button to open the download page. Download the plugin and import it to your project.



## Configure AdMob

After importing Google Mobile Ads plugin, you can now enter the banner and interstitial ad IDs for each platform.



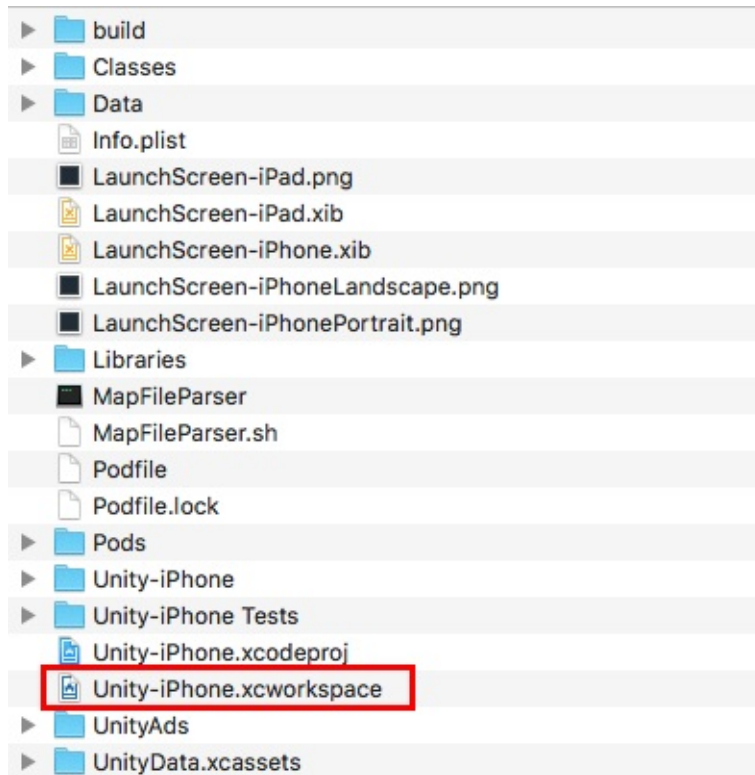
If you only use banner ads you can leave the interstitial ad IDs empty, and vice versa.

If you're not familiar with AdMob, follow the instructions [here](#) to create ad units and obtain the ad IDs; an ad ID should have the form of ca-app-pub-0664570763252260xxxxxxxxxxx.

## Build for iOS

Using Google Mobile Ads on iOS requires [CocoaPods](#) to be installed on your Mac. When building for iOS in Unity, CocoaPods will automatically create an Xcode workspace (with .xcworkspace extension) in the generated Xcode project. You should always open this workspace instead of the normal project file (with .xcodeproj extension).

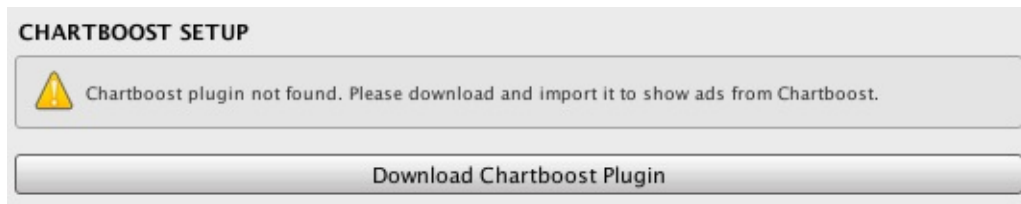




# Chartboost

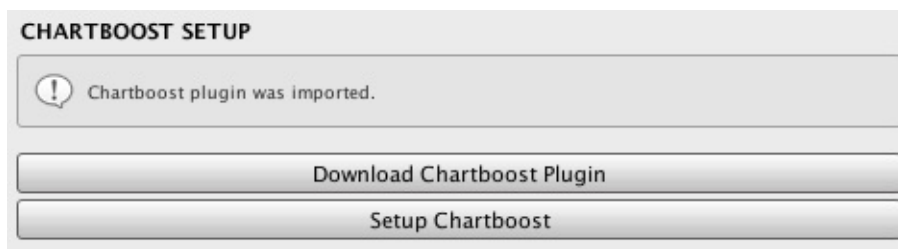
## Import Chartboost Plugin

To show ads from Chartboost you need to import the [Chartboost plugin for Unity](#). In the **CHARTBOOST SETUP** section, click the *Download Chartboost Plugin* button to open the download page. Download the plugin and import it to your project.



## Configure Chartboost

After importing Chartboost plugin, the **CHARTBOOST SETUP** section will be updated as below.



Click the *Setup Chartboost* button to open Chartboost' s settings tool.

The Inspector window displays the ChartboostSettings interface. It includes an 'Open' button in the top right. Below the title bar, there's a text field for 'Add the Chartboost App Id and App Secret associated with this game'. The settings are organized by platform:

- iOS:** App Id [?]: CB\_IOS\_APP\_ID, App Signature [?]: CB\_IOS\_APP\_SIGNATURE
- Google Play:** App Id [?]: CB\_ANDROID\_APP\_ID, App Signature [?]: CB\_ANDROID\_APP\_SIGNATURE
- Amazon:** App Id [?]: CB\_AMAZON\_APP\_ID, App Signature [?]: CB\_AMAZON\_APP\_SIGNATURE

Below these, there's an 'Android Platform [?]:' section with a dropdown menu set to 'Google Play'. Further down, 'Enable Logging for Debug Builds' is shown with an unchecked checkbox labeled 'isLoggingEnabled'. At the bottom, there's a 'Setup Android SDK' button.

Provide the App IDs and App Signatures for your targeted platforms. Remember to click the *Setup Android SDK* button if you're building for Android.

To obtain the App Id and App Signature you need to add your app to the Chartboost dashboard. If you're not familiar with the process please follow the instructions [here](#).

After adding the app, go to APP SETTINGS > Basic Settings to find its App ID and App Signature.

The Chartboost dashboard shows the 'App Settings' page for 'Buster's Boost'. The left sidebar contains navigation links: OVERVIEW, ANALYTICS, CAMPAIGNS, APP SETTINGS (selected), TOOLS, and a 'Delete' button. The 'APP SETTINGS' section is expanded, showing 'Basic Settings'. The form includes:

- Platform:** iOS
- App Nickname:** Buster's Boost
- App Bundle ID:** com.chartboost.apollo
- App Orientation:** Portrait and Landscape (both checked)
- App ID:** 5689b44f8838092b63cf94f0
- App Signature:** edabc1ce4ad0318edd64a4584e59b534c985373c
- Official Name:** Buster's Boost

A large green arrow points from the 'App Bundle ID' field to the 'App ID' and 'App Signature' fields.

## READ\_PHONE\_STATE permission on Android

The Chartboost SDK includes the `READ_PHONE_STATE` permission on Android, to "handle video playback when interrupted by a call", as stated in its manifest. `READ_PHONE_STATE` permission requires your app to have a privacy policy when uploaded to Google Play. Since this permission is not mandatory to run the Chartboost SDK, you can safely remove it if you are not ready to provide the required privacy policy. To remove the permission, open the `AndroidManifest.xml` file located at `Assets/Plugins/Android/ChartboostSDK` folder, then delete the corresponding line (or comment it out as below).

```
<!-- Exclude the READ_PHONE_STATE permission because it requires a privacy policy -->
<!-- <uses-permission android:name="android.permission.android.permission.READ_PHONE_S
TATE" /> -->
```

## Testing Notes

Please note that to show ads from Chartboost you need to either create a publishing campaign or enable the Test Mode for your app.

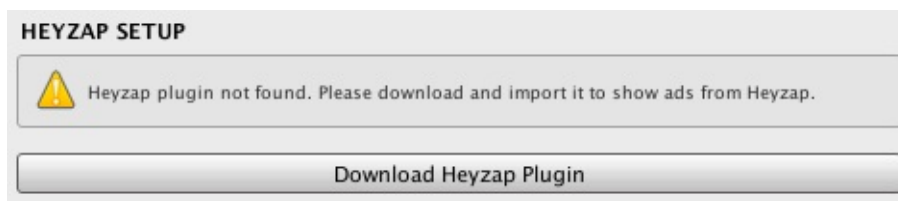
- To create a publishing campaign follow the instructions [here](#)
- To enable Test Mode follow the instruction [here](#)

# Heyzap

Heyzap can serve ads from multiple other networks thanks to its mediation feature. To do so it requires 3rd-party SDKs to be imported together with its own plugin. If you use Heyzap, you should not import the standalone-plugins of other networks (i.e. AdMob, Chartboost and Unity Ads) in order to avoid potential conflicts. Instead, use them as mediated networks under Heyzap and import their corresponding packages provided at the Heyzap download page.


## Import Heyzap Plugin

To show ads from Heyzap you need to import the [Heyzap plugin for Unity](#). In the **CHARTBOOST SETUP** section, click the *Download Heyzap Plugin* button to open the download page.



In the download page select your preferred networks to use with Heyzap mediation. The Heyzap dynamic documentation will update automatically to reflect your selections.

### Network Selection:

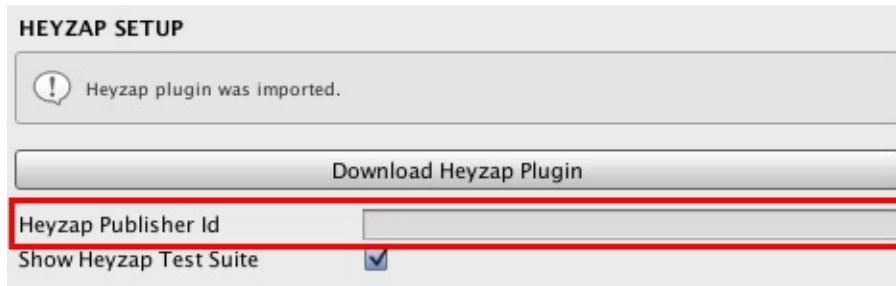
|   |  |   |
|---|--|---|
|  | <input type="checkbox"/> AdColony          | <input checked="" type="checkbox"/> AdMob                     |
| <input checked="" type="checkbox"/> AppLovin  | <input type="checkbox"/> Chartboost        | <input checked="" type="checkbox"/> Facebook Audience Network |
| <input type="checkbox"/> HyprMX   | <input type="checkbox"/> InMobi            | <input type="checkbox"/> Tapjoy                               |
| <input checked="" type="checkbox"/> UnityAds  | <input checked="" type="checkbox"/> Vungle |   |

Follow the instructions provided by Heyzap to download and import its plugin as well as other required 3rd-party plugins. Also go through the **Integration Notes** section below to avoid problems that may occur during the integration of 3rd-party networks.

If you haven't already, use Heyzap's [Integration Wizard](#) to setup the 3rd-party networks to use with mediation.

## Configure Heyzap

After importing Heyzap plugin, the **HEYZAP SETUP** section will be updated as below. You can now enter your publisher Id to the *Heyzap Publisher Id* field.



**HEYZAP SETUP**

Heyzap plugin was imported.

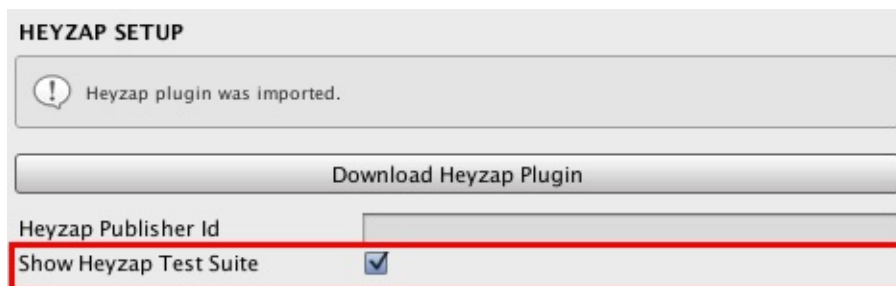
Download Heyzap Plugin

Heyzap Publisher Id

Show Heyzap Test Suite ☒

## Heyzap Mediation Test Suite

The Heyzap plugin comes with a convenient Test Suite that you can use to test each of the networks you selected for mediation. To use this Test Suite, simply check the *Show Heyzap Test Suite* option in the **HEYZAP SETUP** section.



**HEYZAP SETUP**

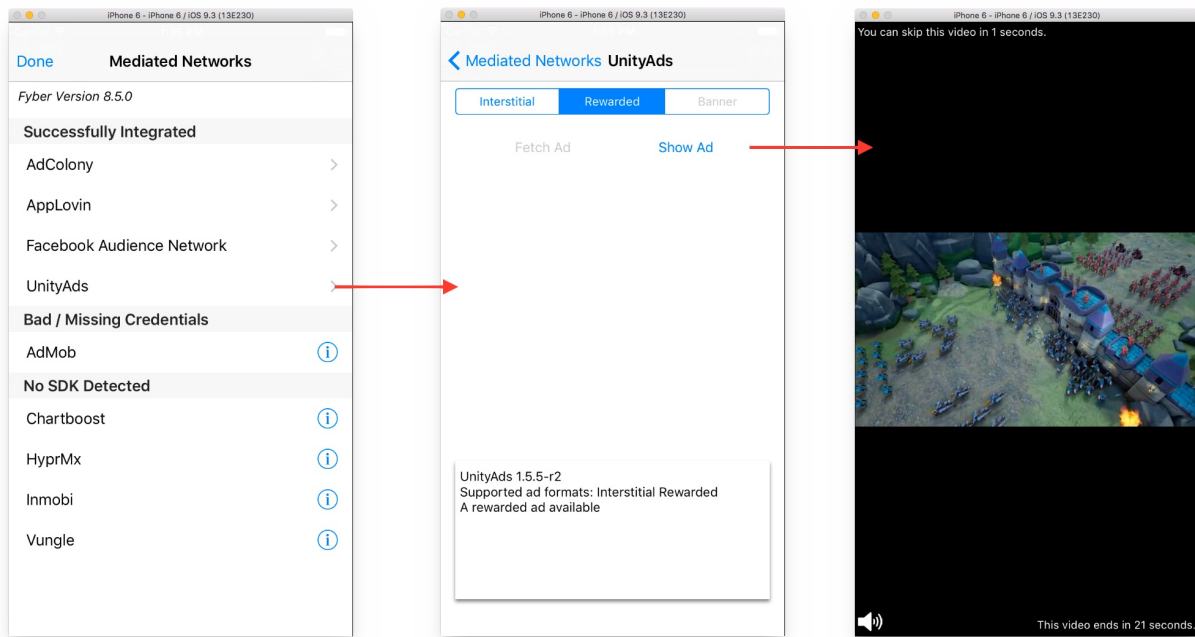
Heyzap plugin was imported.

Download Heyzap Plugin

Heyzap Publisher Id

Show Heyzap Test Suite ☒

Below is the Test Suite interface on iOS.



## Integration Notes

This section discusses some notes that you should take when using Heyzap mediation with various other networks.

### PlayServicesResolver (Android-specific)

As instructed in the Heyzap documentation, the PlayServicesResolver plugin needs to be installed for Android platform. However, if you're using (or intend to use) our Game Service module, you should skip importing this resolver plugin recommended by Heyzap. Instead, import the Google Play Games plugin that comes with our Game Service module. It is accompanied by a compatible version of the PlayServicesResolver, which will help avoid conflicts. Please see the Game Service section for instructions on importing the Google Play Games plugin.

### Facebook Audience Network (Android-specific)

The Facebook Audience Network package contains an *android-support-v4.jar* file under *Assets/Plugins/Android* folder. If your project already contains a *support-v4-xx.x.x.aar* file under that same folder, feel free to remove (or exclude it when importing) the jar file or it will cause the "Unable to convert dex..." error when building due to duplicate libraries.

### AppLovin (Android-specific)

As instructed in the Heyzap documentation, you need to add the AppLovin SDK key to its AndroidManifest.xml file located at *Assets/Plugins/Android/AppLovin* folder. Simply add the following line inside the <application> tag in the manifest, replacing YOUR\_SDK\_KEY with your actual AppLovin SDK key.

```
<meta-data android:name="applovin.sdk.key" android:value="YOUR_SDK_KEY"/>
```

This manifest also includes the `READ_PHONE_STATE` permission, which requires your app to have a privacy policy when uploaded to Google Play. This permission is not mandatory to run the AppLovin SDK, therefore you can safely remove it if you are not ready to provide the required privacy policy. To remove the permission, simply delete the corresponding line from the manifest or comment it out as below.

```
<!-- Exclude the READ_PHONE_STATE permission because it requires a privacy policy -->  
<!-- <uses-permission android:name="android.permission.READ_PHONE_STATE" /> -->
```

### The minSdkVersion Problem (Android-specific)

The current Heyzap SDK requires a `minSdkVersion` of 10, while some other 3rd-party plugins may require a version of 11 or above. If you get a build error including this line

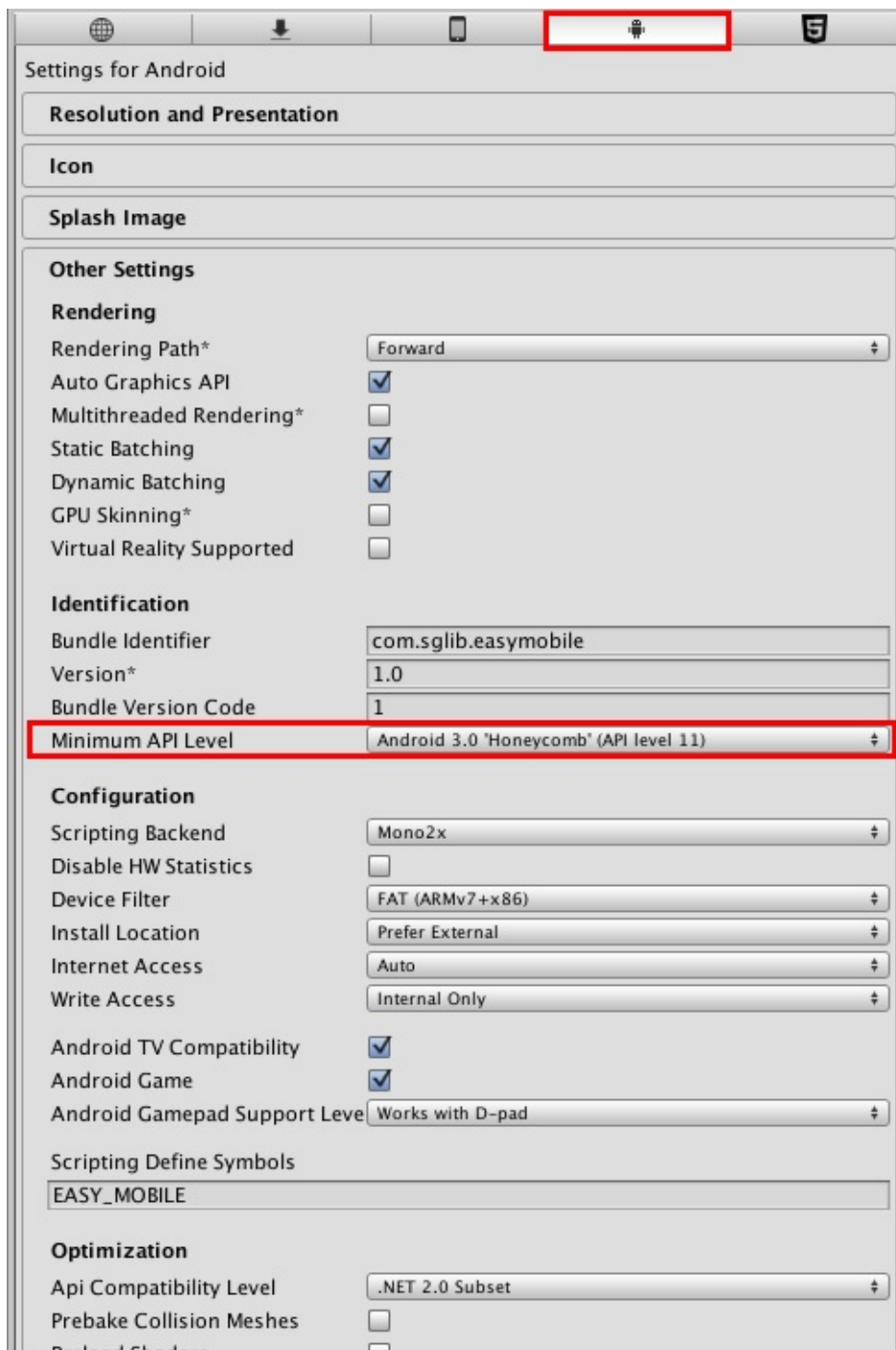
```
Unable to merge android manifests...
```

and this line

```
Main manifest has <uses-sdk android:minSdkVersion='x'> but library uses minSdkVersion=  
'y'
```

where  $x < y$ , it means you need to increase the `minSdkVersion` of the app. To do so go to *Edit > Project Settings > Player*, then select the *Android settings* tab and increase its *Minimum API Level* to the required one (which is 'y' in this example).





# Unity Ads

## Enable Unity Ads Service

To use Unity Ads service, you must first [set up your project for Unity Services](#).

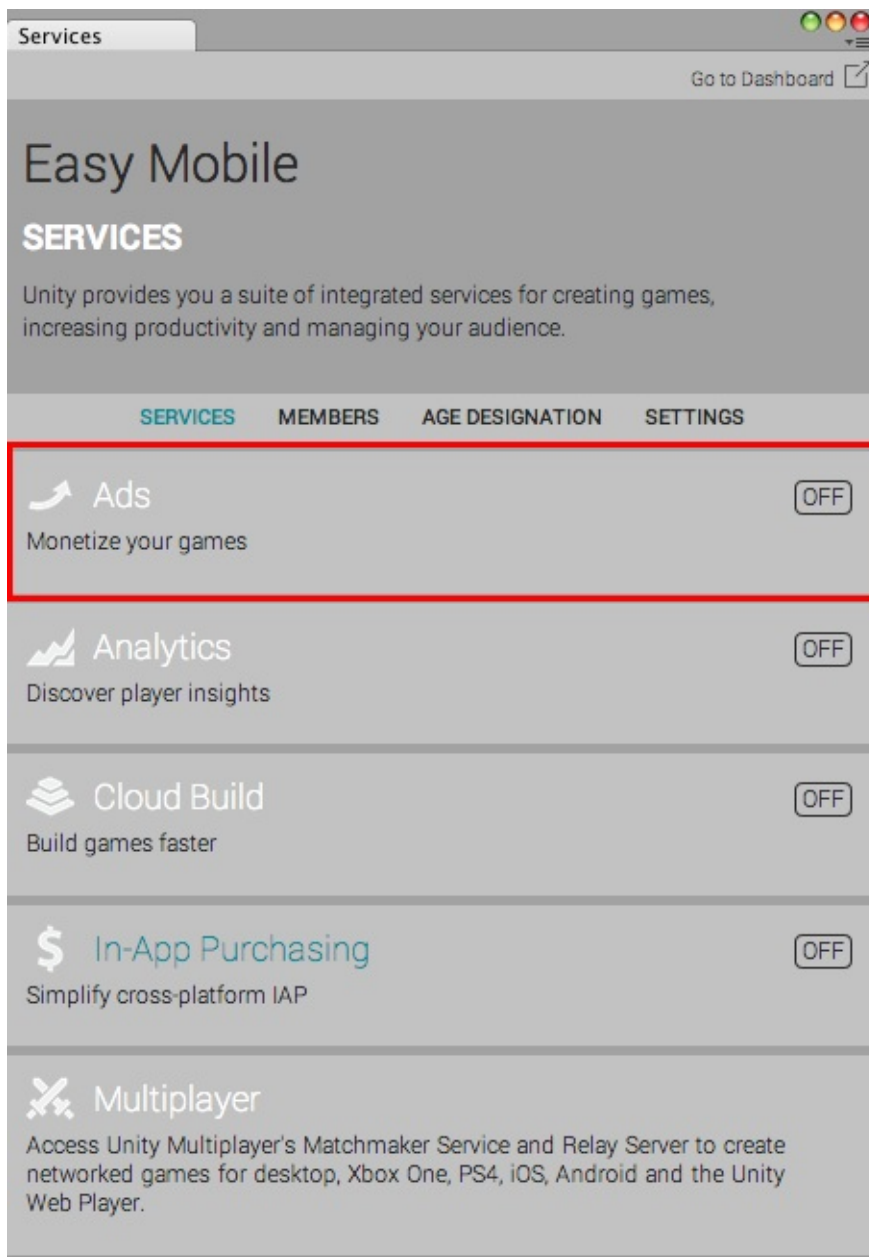
To show ads from Unity Ads you need to enable the corresponding service. Easy Mobile will automatically check for the service's availability and warn you to enable it if needed. Below is the **UNITY ADS SETUP** section when Unity Ads is not enabled.

### UNITY ADS SETUP

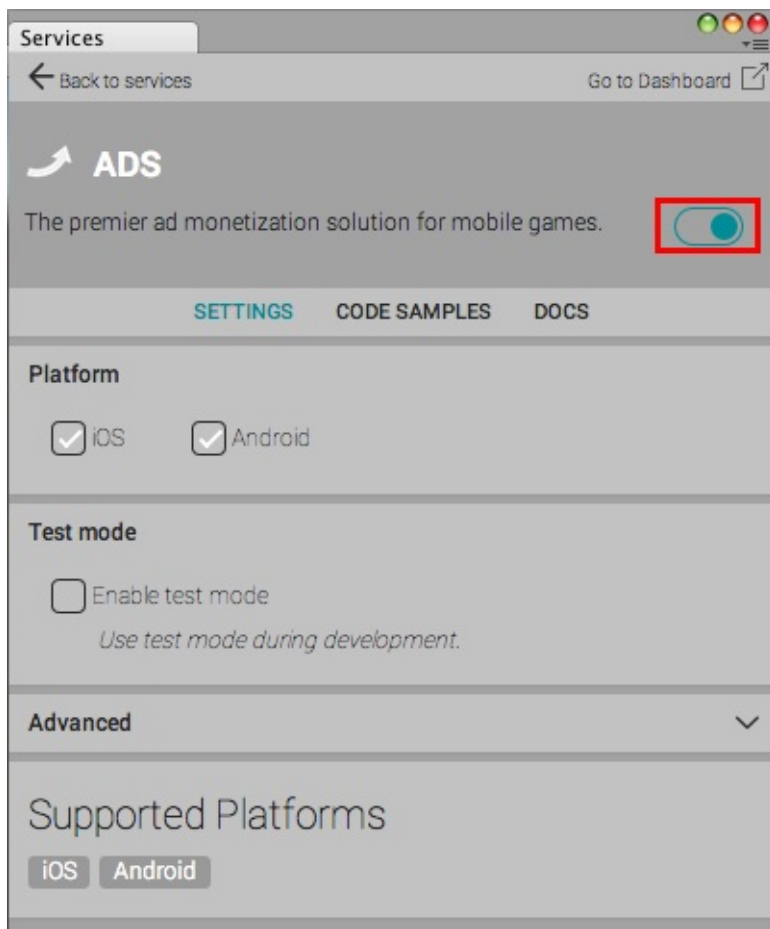


Unity Ads service is disabled or not available for the current platform. To enable it go to Window > Services.

To enable Unity Ads switch the platform to iOS or Android, then go to *Window > Services* and select the Ads tab.



In the opened configuration window, click the toggle at the right-hand side to enable Unity Ads service. You may need to answer a few questions about your game.



The **UNITY ADS SETUP** section will be updated after Unity Ads has been enabled.



## Testing Notes

It is advisable to enable the test mode of Unity Ads during development. This will ensure there's always an ad returned whenever requested. To enable test mode simply check the *Enable test mode* option within the Ads configuration window.

Remember to disable this test mode when creating your release build.

# Automatic Ad Loading

Automatic ad loading is a feature of the Advertising module. It regularly checks for the availability of default ads, and performs loading if needed, to make sure that ads are always ready when they need to be shown. You can configure this feature in the **AUTO AD-LOADING CONFIG** section.

Default ads are ads loaded from default networks, see Default Ad Networks section.

| AUTO AD-LOADING CONFIG |                                     |
|------------------------|-------------------------------------|
| Auto-Load Default Ads  | <input checked="" type="checkbox"/> |
| Ad Checking Interval   | <input type="text" value="10"/>     |
| Ad Loading Interval    | <input type="text" value="20"/>     |

- *Auto-Load Default Ads*: uncheck this to disable automatic ad loading feature, you can load ads manually from script, see **Scripting** section for instructions on this
- *Ad Checking Interval*: change this value to determine how frequently the module should perform ads availability check, the smaller the more frequently
- *Ad Loading Interval*: the minimum time between two ad loading requests

## Default Ad Networks

You can select default ad networks for each platform in the **DEFAULT AD NETWORKS** section. You can have different networks for different ad types and different selections for different platforms. If you don't want to use a certain type of ad, simply set its network to *None*.

**DEFAULT AD NETWORKS**

▼ [iOS] Default Ad Networks

|                         |            |
|-------------------------|------------|
| Banner Ad Network       | Ad Mob     |
| Interstitial Ad Network | Chartboost |
| Rewarded Ad Network     | Unity Ads  |

▼ [Android] Default Ad Networks

|                         |           |
|-------------------------|-----------|
| Banner Ad Network       | Ad Mob    |
| Interstitial Ad Network | Ad Mob    |
| Rewarded Ad Network     | Unity Ads |

⚠ Default ad network AdMob has no SDK. Please import its plugin.

⚠ Default ad network Chartboost has no SDK. Please import its plugin.

⚠ Default ad network UnityAds has no SDK. Please import its plugin.

Pay attention to the warnings and import the required plugins if you haven't already.

# Scripting

This section provides a guide to work with the Advertising API using the default ad networks configured in the module settings.

You can access all the Advertising API methods via the `AdManager` class under the `EasyMobile` namespace.

## Banner Ads

To show a banner ad you need to specify its position using the *BannerAdPosition* enum. The banner will be displayed once it is loaded.

```
// Show banner ad
AdManager.ShowBannerAd(BannerAdPosition.Bottom);
```

To hide the current banner ad (it can be shown again later):

```
// Hide banner ad
AdManager.HideBannerAd();
```

To destroy the current banner ad (a new one will be created on the next banner ad showing):

```
// Destroy banner ad
AdManager.DestroyBannerAd();
```

You can also check if a banner ad is being shown:

```
// Check if banner ad is being shown
bool isShowingBanner = AdManager.IsShowingBannerAd();
```

## Interstitial Ads

The method to show an interstitial ad requires it to be already loaded. Therefore you should check for the ad's availability before showing it.

```
// Check if interstitial ad is ready
bool isReady = AdManager.IsInterstitialAdReady();

// Show it if it's ready
if (isReady)
{
    AdManager.ShowInterstitialAd();
}
```

An *InterstitialAdCompleted* event will be fired whenever an interstitial ad is closed. You can listen to this event to take appropriate actions, e.g. resume the game.

```
// Subscribe to the event
void OnEnable()
{
    AdManager.InterstitialAdCompleted += InterstitialAdCompletedHandler;
}

// The event handler
void InterstitialAdCompletedHandler(InterstitialAdNetwork network, AdLocation location)
{
    Debug.Log("Interstitial ad has been closed.");
}

// Unsubscribe
void OnDisable()
{
    AdManager.InterstitialAdCompleted -= InterstitialAdCompletedHandler;
}
```

## Rewarded Ads

The method to show a rewarded ad requires it to be already loaded. Therefore you should check for the ad's availability before showing it.

```
// Check if rewarded ad is ready
bool isReady = AdManager.IsRewardedAdReady();

// Show it if it's ready
if (isReady)
{
    AdManager.ShowRewardedAd();
}
```

A *RewardedAdCompleted* event will be fired whenever a rewarded ad has completed. You should listen to this event to reward the user for watching the ad.



```
// Subscribe to the event
void OnEnable()
{
    AdManager.RewardedAdCompleted += RewardedAdCompletedHandler;
}

// The event handler
void RewardedAdCompletedHandler(RewardedAdNetwork network, AdLocation location)
{
    Debug.Log("Rewarded ad has completed. The user should be rewarded now.");
}

// Unsubscribe
void OnDisable()
{
    AdManager.RewardedAdCompleted -= RewardedAdCompletedHandler;
}
```

## Remove Ads

In some cases you need to remove/stop showing ads in your game, e.g. when the user purchases the "Remove Ads" product. To remove ads:

```
// Remove ads permanently
AdManager.RemoveAds();
```

The *RemoveAds* method will destroy the banner ad if one is being shown, and prevent future ads from being loaded and shown except rewarded ads.

Rewarded ads can still be shown after ads were removed since they are unobtrusive and only shown at the user discretion.

An *AdsRemoved* event will be fired after ads have been removed. You can listen to this event and take appropriate actions, e.g update the UI.

```
// Subscribe to the event
void OnEnable()
{
    AdManager.AdsRemoved += AdsRemovedHandler;
}

// The event handler
void AdsRemovedHandler()
{
    Debug.Log("Ads were removed.");

    // Unsubscribe
    AdManager.AdsRemoved -= AdsRemovedHandler;
}
```

You can also check at any time if ads were removed or not.

```
// Determine if ads were removed
bool isRemoved = AdManager.IsAdRemoved();
```

Finally, you can also revoke the ad removing status and allow ads to be shown again.

```
// Revoke ad removing status and allow showing ads again
AdManager.ResetRemoveAds();
```

## Manual Ad Loading

Normally you don't need to worry about loading ads if the automatic ad loading feature is enabled (see **Configure Advertising Module** section). Otherwise, if you choose to disable this feature, you can load ads manually from script.

It is advisable to load an ad as far in advance of showing it as possible to allow time for the ad to be loaded.

To load an interstitial ad:

```
// Load an interstitial ad
AdManager.LoadInterstitialAd();
```

To load a rewarded ad:

```
// Load a rewarded ad
AdManager.LoadRewardedAd();
```



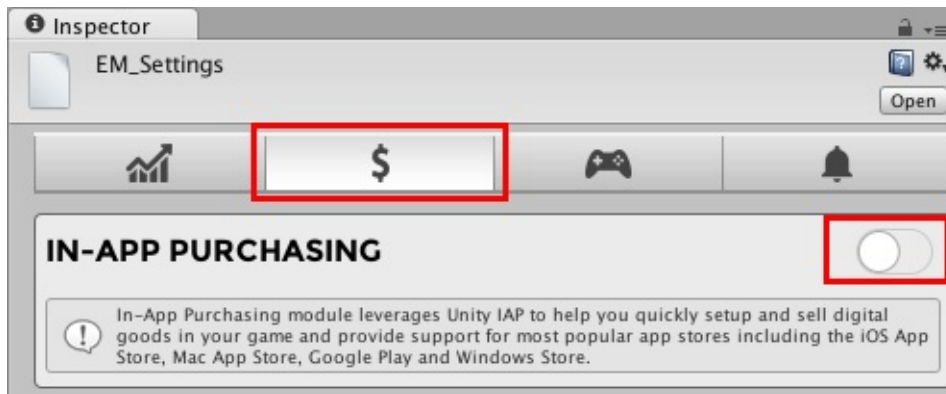
# In-App Purchasing

The In-App Purchasing module helps you quickly setup and sell digital products in your game. Here're some highlights of this modules:

- **Leverages Unity In-App Purchasing service**
  - This module is built on top of Unity IAP service, a powerful service that supports most app stores including iOS App Store, Google Play, Amazon Apps, Samsung GALAXY Apps and Tizen Store
  - Unity IAP is tightly integrated with the Unity engine, so compatibility and reliability can be expected
- **Easy management of product catalog**
  - Easy Mobile's custom editor features a friendly interface that helps you easily add, edit or remove products
- **Receipt validation**
  - Local receipt validation that offers extra security

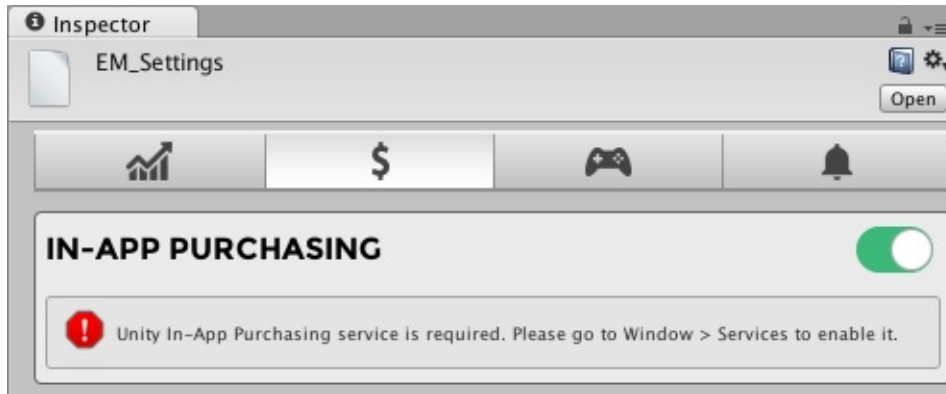
# Module Configuration

To use the In-App Purchasing module you must first enable it. Go to *Window > Easy Mobile > Settings*, select the In-App Purchasing tab, then click the right-hand side toggle to enable and start configuring the module.



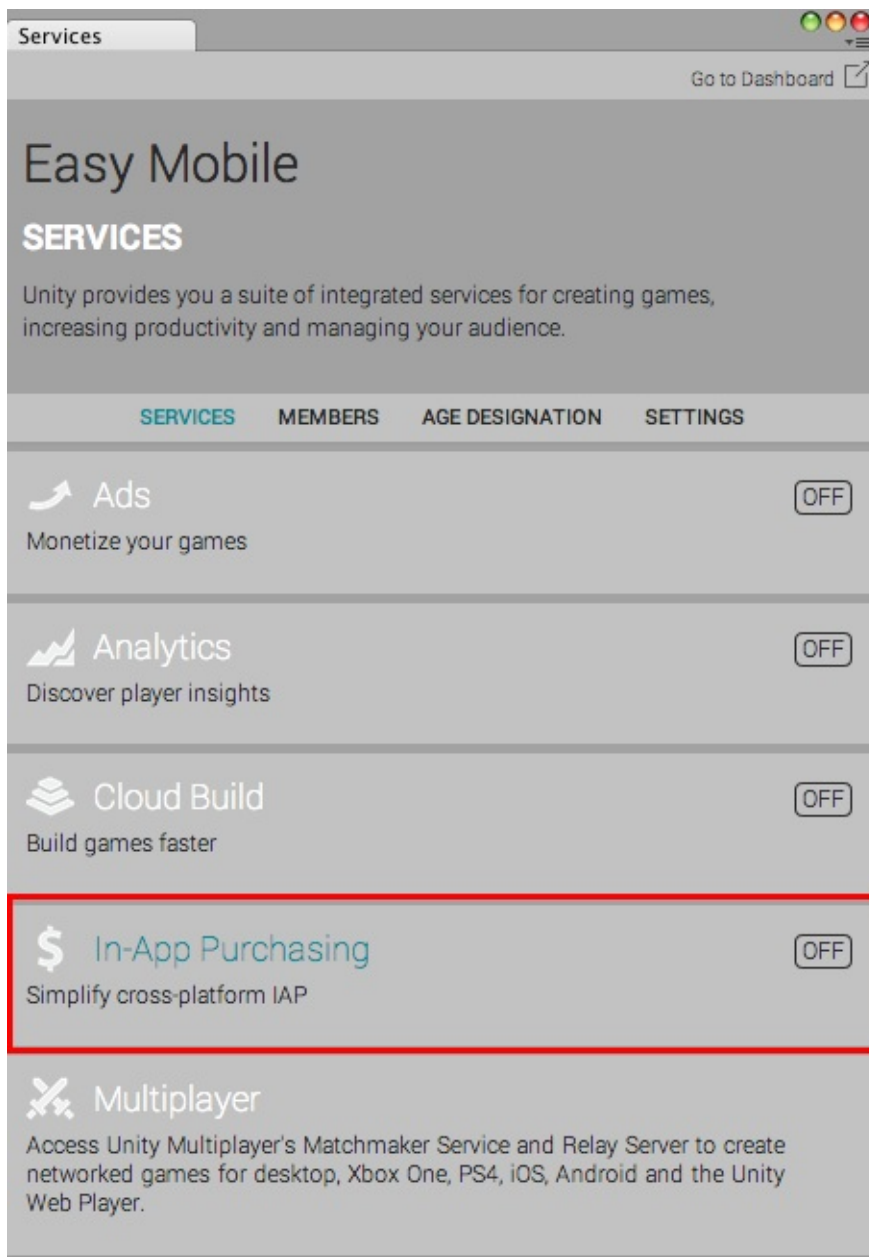
# Enable Unity IAP

The In-App Purchasing module requires Unity IAP service to be enabled. It will automatically check for the service's availability and prompt you to enable it if needed. Below is the module settings interface when Unity IAP is disabled.

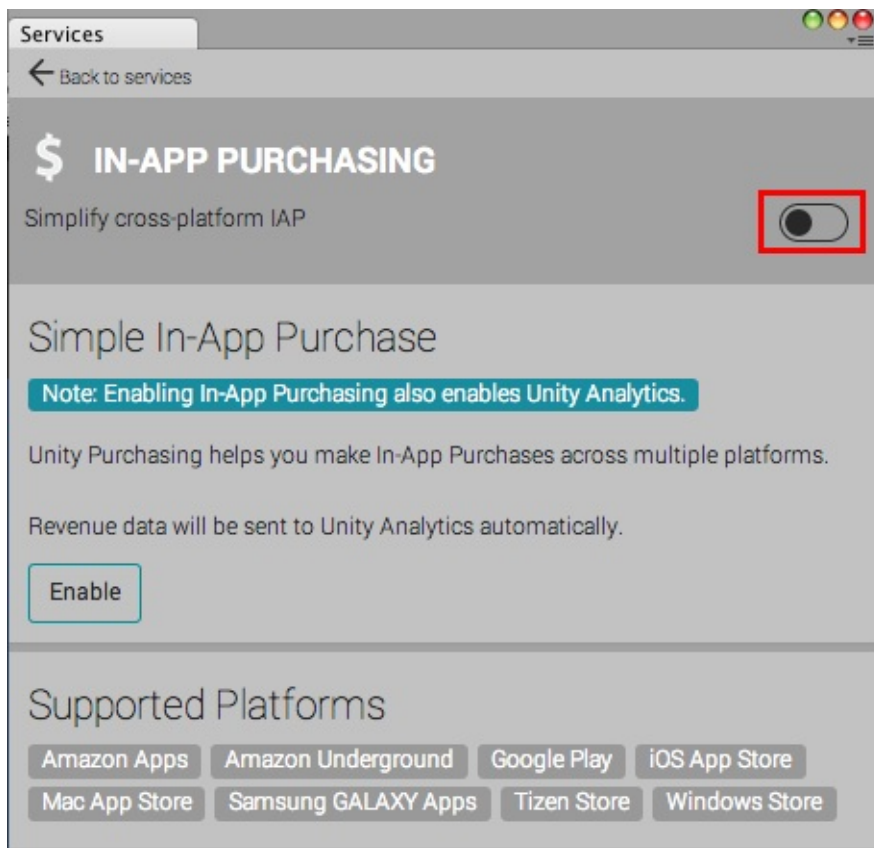


To use Unity In-App Purchasing service, you must first [set up your project for Unity Services](#).

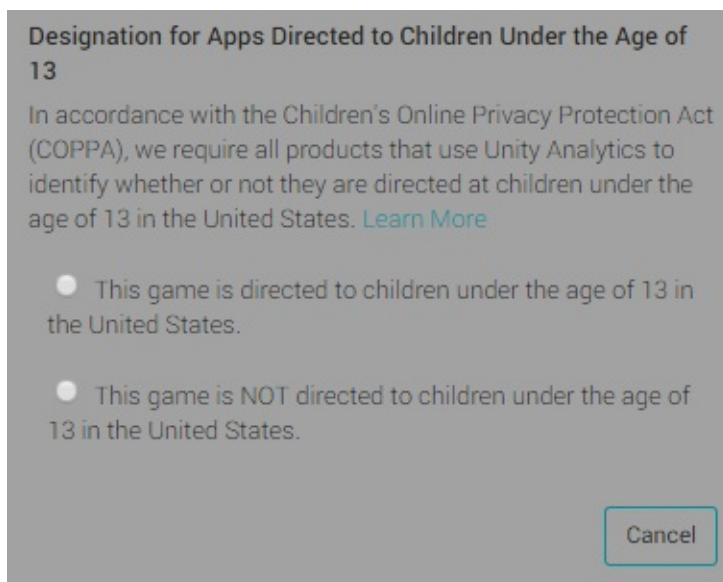
To enable Unity IAP service go to *Window > Services* and select the In-App Purchasing tab.



In the opened configuration window, click the toggle at the right-hand side or the *Enable* button to enable Unity IAP service.

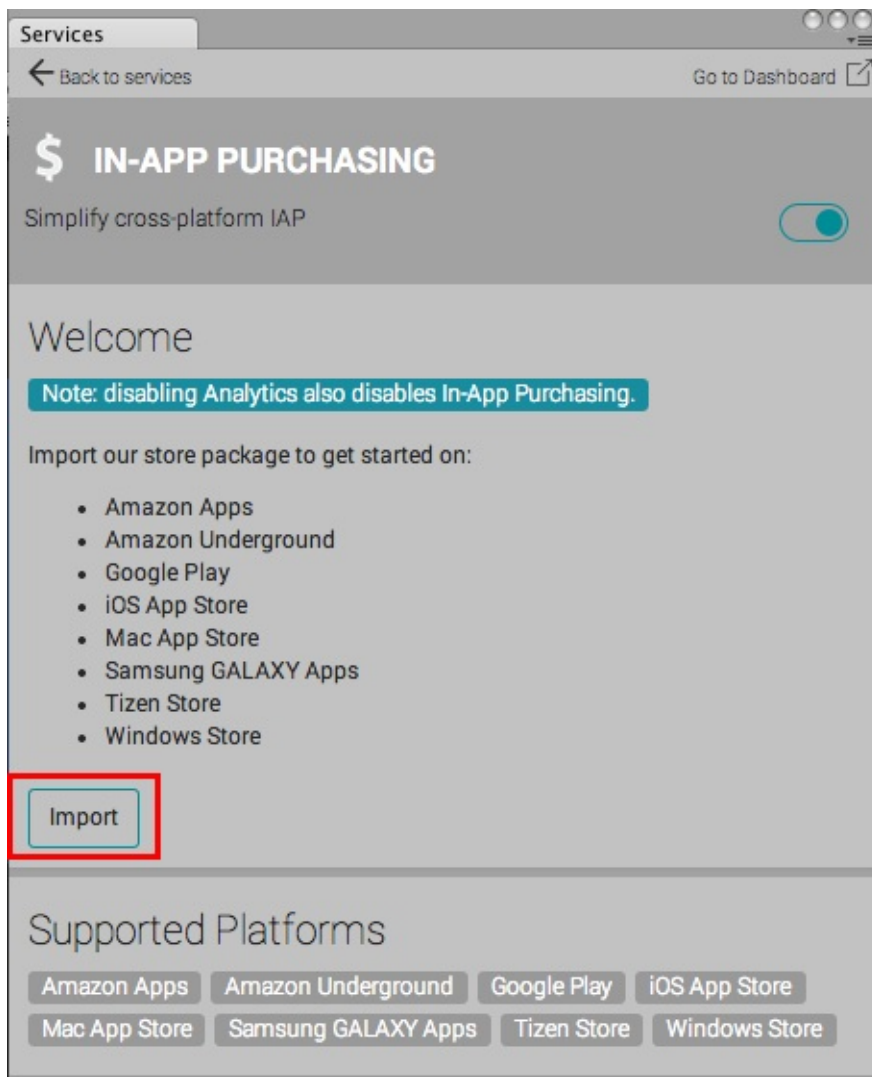


A dialog window will appear asking a few questions about your game in order to ensure COPPA compliance.



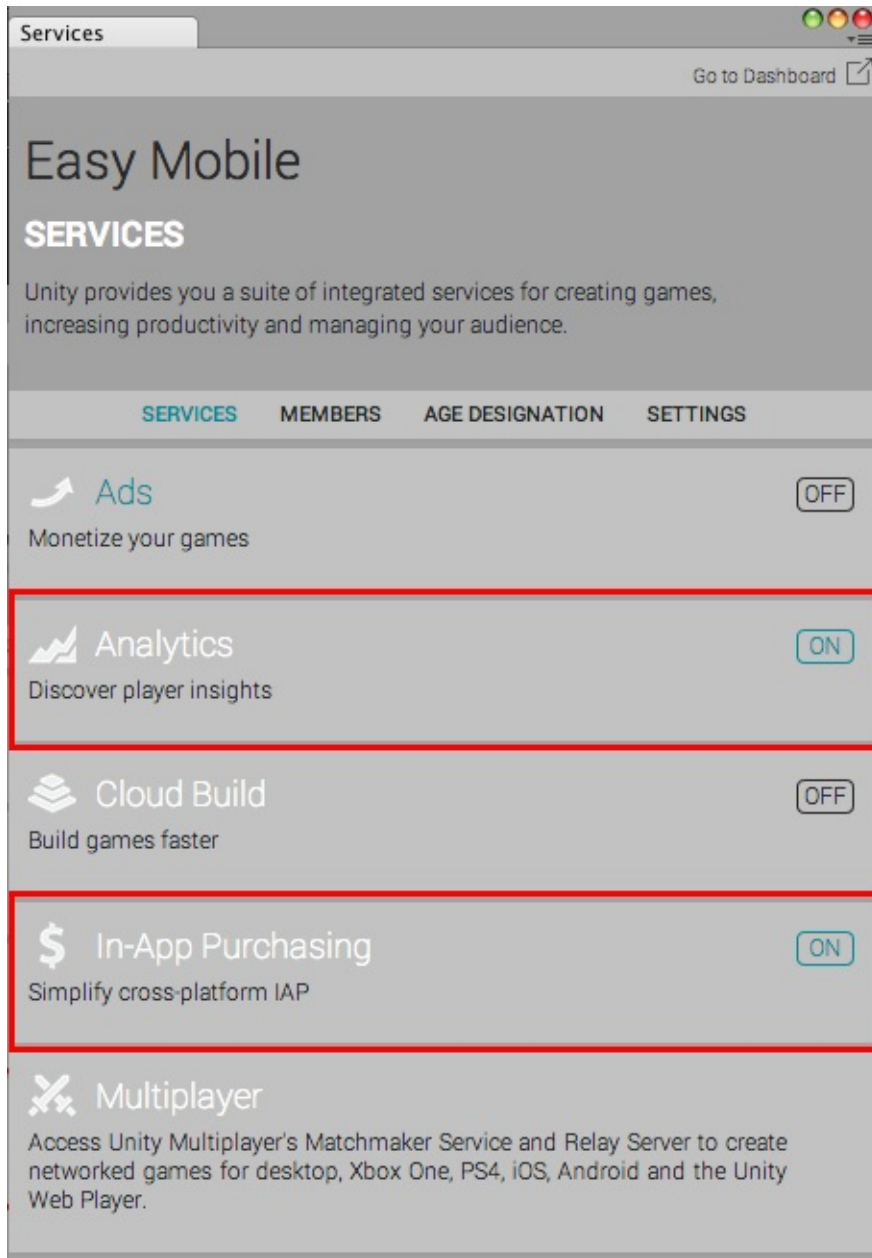
Next click the *Import* button to import the Unity IAP package to your project.



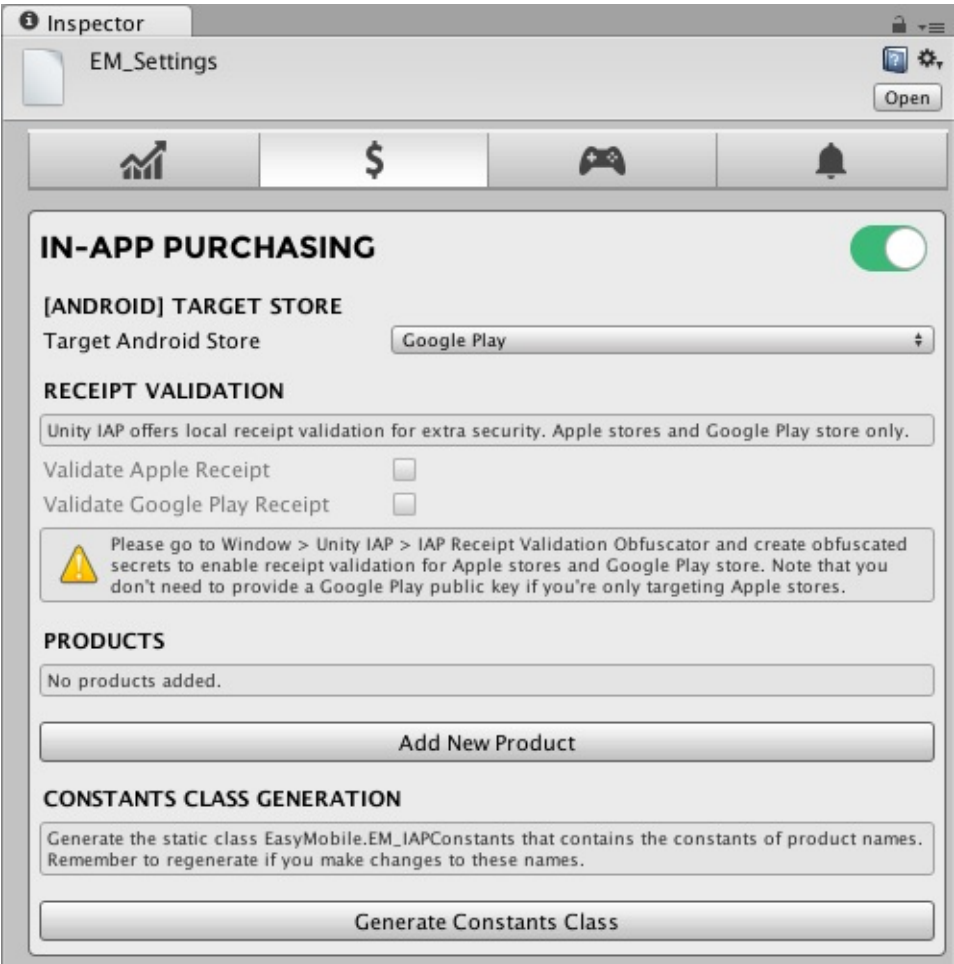


After importing, there should be a UnityPurchasing folder added under Assets/Plugins folder.

Enabling Unity IAP service will automatically enable the Unity Analytics service (if it's not enabled before), this is a requirement to use Unity IAP. Go back to the Services panel and make sure that both In-App Purchasing and Analytics services are now enabled.

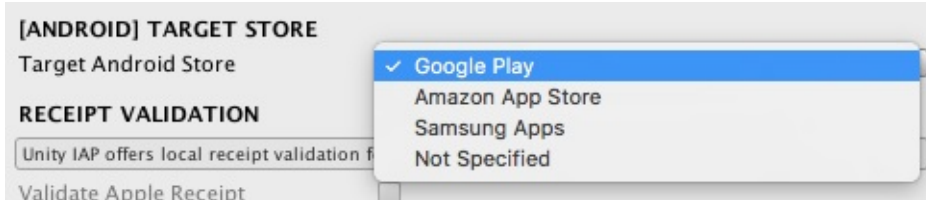


After enabling Unity IAP service, the settings interface of the In-App Purchasing module will be updated and ready for you to start configuring.



# Target Android Store

If you're building for Android platform, you need to specify your target store. In the **[ANDROID] TARGET STORE** section select your target store from the dropdown.



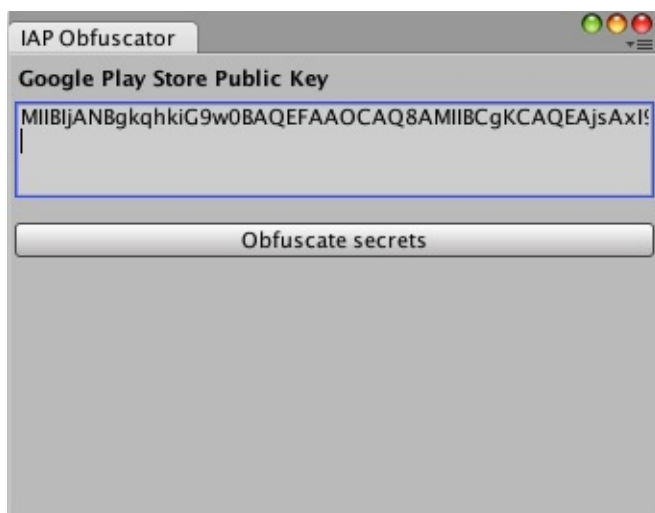
# Receipt Validation

The receipt validation feature provides extra security and helps prevent prevent fraudulent users from accessing content they have not purchased. This feature employs Unity IAP's local receipt validation, which means the validation takes place on the target device, without the need to connect to a remote server.

Receipt validation is available for Apple stores and Google Play store. Please find more information about Unity IAP's receipt validation [here](#).

## Obfuscating Encryption Keys

To enable receipt validation, you must first create obfuscated encryption keys. The purpose of this obfuscating process is to prevent a fraudulent user from accessing the actual keys, which are used for the validation process. To obfuscate your encryption keys, go to *Window > Unity IAP > Receipt Validation Obfuscator*.



In the opened **IAP Obfuscator** window, paste in your Google Play public key and hit the *Obfuscate secrets* button. According to Unity documentation, this will obfuscate both Apple's root certificate (bundle with Unity IAP) and the provided Google Play public key and create two C# files *AppleTangle* and *GooglePlayTangle* at *Assets/Plugins/UnityPurchasing/generated*. These files are required for the receipt validation process.

To obtain the Google Play public key for your app, login to your Google Play Developer Console, select your app, then navigate to the **Services & APIs** section and find your key under the section labeled **YOUR LICENSE KEY FOR THIS APPLICATION**.

Note that you don't need to provide a Google Play public key if you're only targeting Apple stores.

## Enable Receipt Validation

After creating the obfuscated encryption keys, you can now enable receipt validation for your game. Open the In-App Purchasing module settings, then in the **RECEIPT VALIDATION** section check the corresponding options for your targeted stores.

**RECEIPT VALIDATION**

Unity IAP offers local receipt validation for extra security. Apple stores and Google Play store only.

Validate Apple Receipt

☒

Validate Google Play Receipt

☒

# Product Management

In the **PRODUCTS** section you can easily add, edit or remove your IAP products.

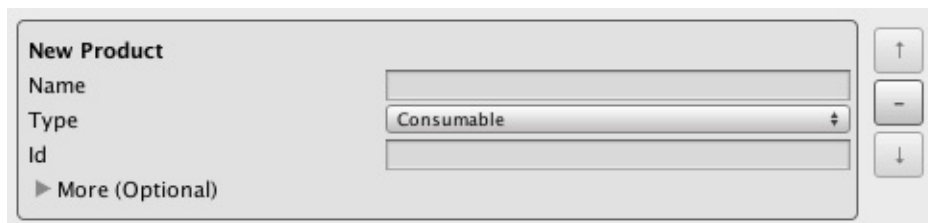
## Add a New Product

To add a new product, click the *Add New Product* button.



The screenshot shows a grey header bar with the word "PRODUCTS" in bold. Below it is a light grey box containing the text "No products added." and a button labeled "Add New Product".

A new empty product will be added.

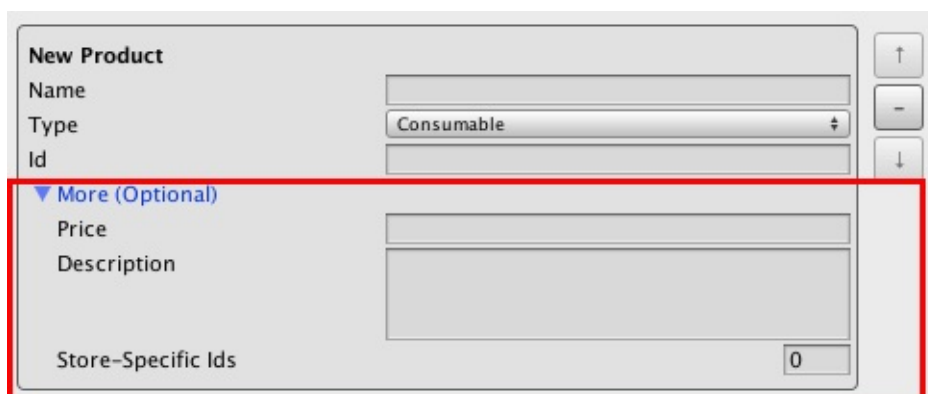


The screenshot shows a "New Product" form. It has three input fields: "Name", "Type" (which is a dropdown menu currently showing "Consumable"), and "Id". Below these fields is a link that says "More (Optional)". To the right of the form are three buttons: an up arrow, a minus sign, and a down arrow.

Fill in the required information for your new product:

- *Name*: the product name, can be used when making purchases
- *Type*: the product type, can be Consumable, Non-Consumable or Subscription
- *Id*: the unified product identifier, you should use this ID when declaring the product on your targeted stores; otherwise, if you need to have a different ID for this product on a certain store, add it to the *Store-Specific Ids* array (see below)

Click *More* if you need to enter store-specific IDs or fill in optional information for your product.



This screenshot shows the "New Product" form with the "More (Optional)" section expanded. This section contains three fields: "Price", "Description" (a larger text area), and "Store-Specific Ids" (a small input field with the value "0"). A red rectangle highlights the entire "More (Optional)" section.

- *Price*: the product price string for displaying purpose
- *Description*: the product description for displaying purpose

- *Store-Specific Ids*: if you need to use a different product ID (than the unified ID provided above) on a certain store, you can add it here

### Adding Store-Specific ID

To add a new ID to the *Store-Specific Ids* array, increase the array size by adjusting the number in the right-hand side box. A new record will be added where you can select the targeted store and enter the corresponding product ID for that store.

Below is a sample product with all the information entered including the two store-specific IDs.

The screenshot shows a form titled "Sample Product". It contains the following fields and values:

- Name: Sample Product
- Type: Consumable
- Id: com.easymobile.sample\_product
- More (Optional) section:
  - Price: \$1.99
  - Description: This is sample consumable product
- Store-Specific Ids section:
  - A dropdown menu with "Apple App Store" selected.
  - A dropdown menu with "Amazon Apps" selected.
  - A text input field with "com.easymobile.sample\_product\_ios" for Apple and "com.easymobile.sample\_product\_amazon" for Amazon.
  - A counter box on the right showing the number "2".

On the right side of the form, there are three buttons: an up arrow, a minus sign, and a down arrow.

## Remove a Product

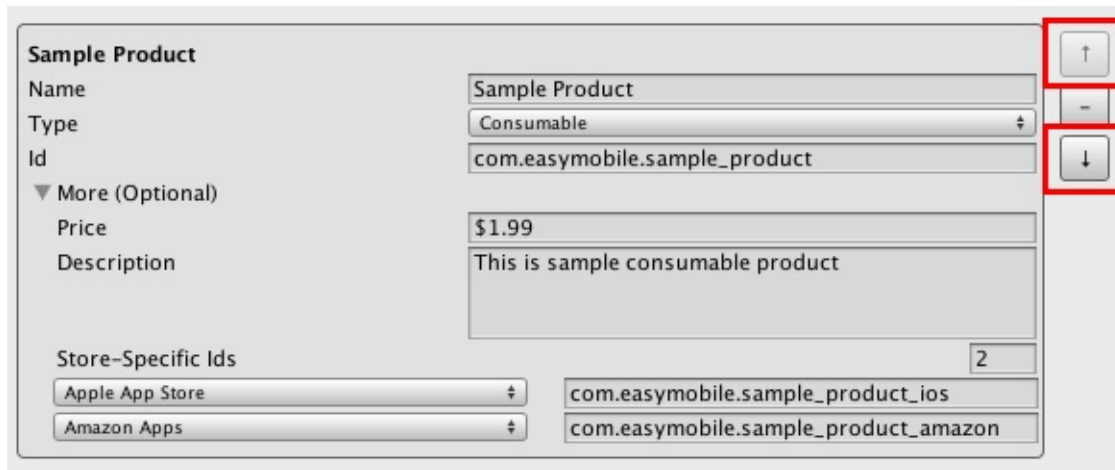
To remove a product, simply click the [-] button at the right hand side.

This screenshot is identical to the one above, but the minus button (-) on the right side of the form is highlighted with a red rectangle, indicating it is the button used to remove the product.

## Arrange Product List

You can use the two arrow-up and arrow-down buttons to move a product upward or downward within the product list.





The screenshot shows the 'Sample Product' configuration window in Unity. The form includes fields for Name, Type, Id, Price, and Description. The 'More (Optional)' section is expanded, showing 'Store-Specific Ids' with a count of 2. Below this, there are two rows for different stores: 'Apple App Store' and 'Amazon Apps', each with a corresponding ID field. On the right side of the form, there are three buttons: an up arrow, a minus sign, and a down arrow. The up and down arrow buttons are highlighted with red boxes.

|                    |                                      |
|--------------------|--------------------------------------|
| Name               | Sample Product                       |
| Type               | Consumable                           |
| Id                 | com.easymobile.sample_product        |
| Price              | \$1.99                               |
| Description        | This is sample consumable product    |
| Store-Specific Ids | 2                                    |
| Apple App Store    | com.easymobile.sample_product_ios    |
| Amazon Apps        | com.easymobile.sample_product_amazon |

## Setup Products for Targeted Stores

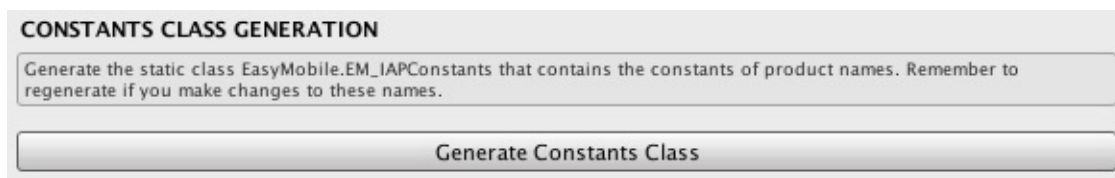
Beside creating the product list in Unity, you also need to declare similar products for your targeted stores, e.g. if you're targeting iOS App Store you need to create the products in iTunes Connect. If you're not familiar with the process, you can follow [Unity's instructions on configuring IAP for various stores](#), which also include useful information about IAP testing.

On Google Play store, both consumable and non-consumable products are defined as Managed product. If a product is set to Consumable type in Unity, the module will automatically handle the consumption of the product once it is bought and make it available for purchase again.

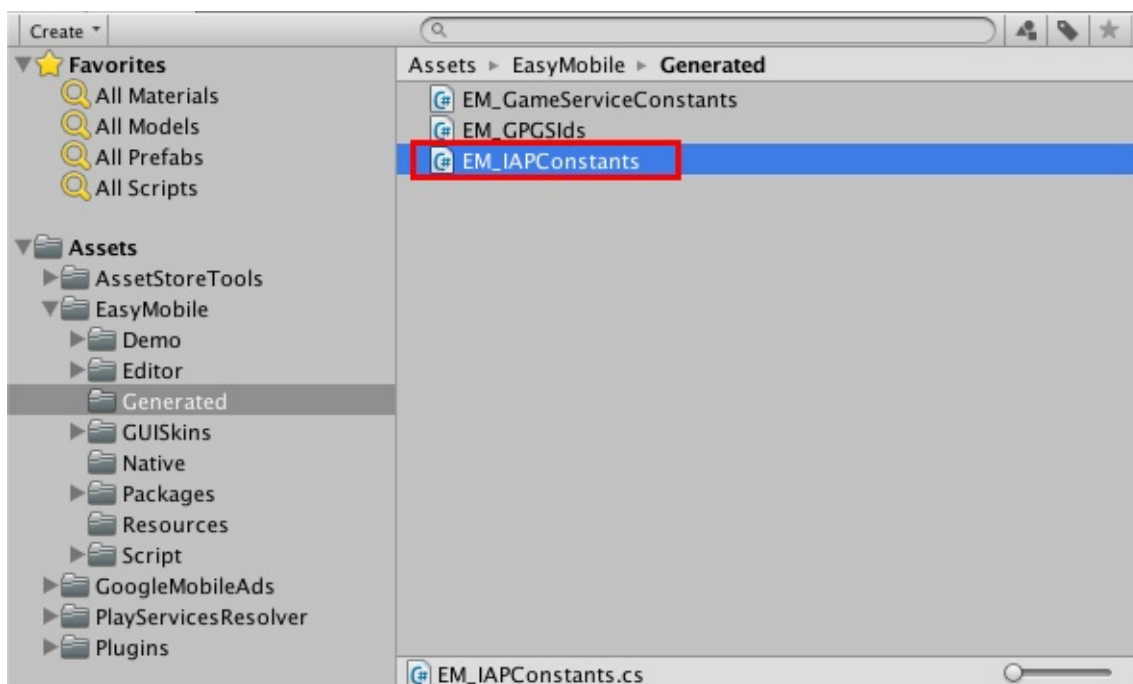
# IAP Constants Generation

Constants generation is a feature of the In-App Purchasing module. It reads all the product names and generates a static class named `EM_IAPConstants` that contains the constants of these names. Later, you can use these constants when making purchases in script instead of typing the product names directly, thus help prevent runtime errors due to typos and the likes.

To generate the constants class (you should do this after finishing with product editing), click the *Generate Constants Class* button within the **CONSTANTS CLASS GENERATION** section.



When the process completes, a file named `EM_IAPConstants` will be created at *Assets/EasyMobile/Generated*.



# Scripting

This section provides a guide to work with the In-App Purchasing API.

You can access all the In-App Purchasing API methods via the `IAPManager` class under the `EasyMobile` namespace.

## Initialization

The module will automatically initialize Unity IAP at start without you having to do anything. All further API calls can only be made after the initialization has finished. You can check if Unity IAP has been initialized:

```
// Check if Unity IAP has been initialized
bool isInitialized = IAPManager.IsInitialized();
```

## Obtain Product List

You can obtain an array of all products created in the module settings interface:

```
// Get the array of all products created in the In-App Purchasing module settings
// IAPProduct is the class representing a product as declared in the module settings
// The InAppPurchasing property of EM_Settings class holds the settings of this module
IAPProduct[] products = EM_Settings.InAppPurchasing.Products;

// Print all product names
foreach (IAPProduct prod in products)
{
    Debug.Log("Product name: " + prod.Name);
}
```

## Make a Purchase

You can purchase a product using its name.

It is strongly recommended that you use the constants of product names in the generated `EM_IAPConstants` class (see **IAP Constants Generation** section) instead of typing the names directly in order to prevent runtime errors due to typos and the likes.

```
// Purchase a product using its name
// EM_IAPConstants.Sample_Product is the generated name constant of a product named "Sample Product"
IAPManager.Purchase(EM_IAPConstants.Sample_Product);
```

A *PurchaseCompleted* event will be fired if the purchase is successful, otherwise, a *PurchaseFailed* event will be fired instead. You can listen to these events and take appropriate actions, e.g. grant the user digital goods if the purchase has succeeded.

```
// Subscribe to IAP purchase events
void OnEnable()
{
    IAPManager.PurchaseCompleted += PurchaseCompletedHandler;
    IAPManager.PurchaseFailed += PurchaseFailedHandler;
}

// Successful purchase handler
void PurchaseCompletedHandler(IAPProduct product)
{
    // Compare product name to the generated name constants to determine which product was bought
    switch (product.Name)
    {
        case EM_IAPConstants.Sample_Product:
            Debug.Log("Sample_Product was purchased. The user should be granted it now.");
            break;
        case EM_IAPConstants.Another_Sample_Product:
            Debug.Log("Another_Sample_Product was purchased. The user should be granted it now.");
            break;
        // More products here...
    }
}

// Failed purchase handler
void PurchaseFailedHandler(IAPProduct product)
{
    Debug.Log("The purchase of product " + product.Name + " has failed.");
}

// Unsubscribe
void OnDisable()
{
    IAPManager.PurchaseCompleted -= PurchaseCompletedHandler;
    IAPManager.PurchaseFailed -= PurchaseFailedHandler;
}
```

## Check for Product Ownership

You can check if a product is owned by specifying its name. A product is considered "owned" if its receipt exists and passes the receipt validation (if enabled).

```
// Check if the product is owned by the user
// EM_IAPConstants.Sample_Product is the generated name constant of a product named "S
ample Product"
bool isOwned = IAPManager.IsProductOwned(EM_IAPConstants.Sample_Product);
```

Consumable products' receipts are not persisted between app restarts, therefore this method only returns true for those products in the session they're purchased.

## Restore Purchases

Non-consumable and subscription products are restorable. App stores maintain a permanent record of each user's non-consumable and subscription products, so that he or she can be granted these products again when reinstalling your game.

Apple normally requires a *Restore Purchases* button to exist in your game, so that the users can explicitly initiate the purchase restoration process. On other platforms, e.g. Google Play, the restoration is done automatically during the first initialization after reinstallation.

During the restoration process, a *PurchaseCompleted* event will be fired for each owned product, as if the user has just purchased them again. Therefore you can reuse the same handler to grant the user their products as normal purchases.

On Apple platforms, you can initiate a purchase restoration as below.

```
// Restore purchases. This method only has effect on Apple platforms.
IAPManager.RestorePurchases();
```

A *RestoreCompleted* event will be fired if the restoration is successful, otherwise, a *RestoreFailed* event will be fired instead. Note that these events only mean the success or failure of the restoration itself, while the *PurchaseCompleted* event will be fired for each restored product, as noted earlier. You can listen to these events and take appropriate actions, e.g. inform the user the restoration result.

```
// Subscribe to IAP restore events, these events are fired on Apple platforms only
void OnEnable()
{
    IAPManager.RestoreCompleted += RestoreCompletedHandler;
    IAPManager.RestoreFailed += RestoreFailedHandler;
}

// Successful restoration handler
void RestoreCompletedHandler()
{
    Debug.Log("All purchases have been restored successfully.");
}

// Failed restoration handler
void RestoreFailedHandler()
{
    Debug.Log("The purchase restoration has failed.");
}

// Unsubscribe
void OnDisable()
{
    IAPManager.RestoreCompleted -= RestoreCompletedHandler;
    IAPManager.RestoreFailed -= RestoreFailedHandler;
}
```

## Retrieve Product Localized Data

You can get a product's metadata retrieved from targeted app stores, e.g. localized title, description and price. This information is particularly useful in building a storefront in your game for displaying the in-app products. To get the localized data of a product, simply specify its name:

```
// Get product localized data retrieved from the targeted app store.
// EM_IAPConstants.Sample_Product is the generated name constant of a product named "Sample Product"
ProductMetadata data = GetProductLocalizedData(EM_IAPConstants.Sample_Product);
```

The following example loops through the product list and retrieve the localized data of each item.

```
// Get the array of all products created in the In-App Purchasing module settings
IAPProduct[] products = EM_Settings.InAppPurchasing.Products;

// Print products' localized data
foreach (IAPProduct prod in products)
{
    ProductMetadata data = GetProductLocalizedData(prod.Name);

    if (data != null)
    {
        Debug.Log("Localized title: " + data.localizedTitle);
        Debug.Log("Localized description: " + data.localizedDescription);
        Debug.Log("Localized price string: " + data.localizedPriceString);
    }
}
```

# Game Service

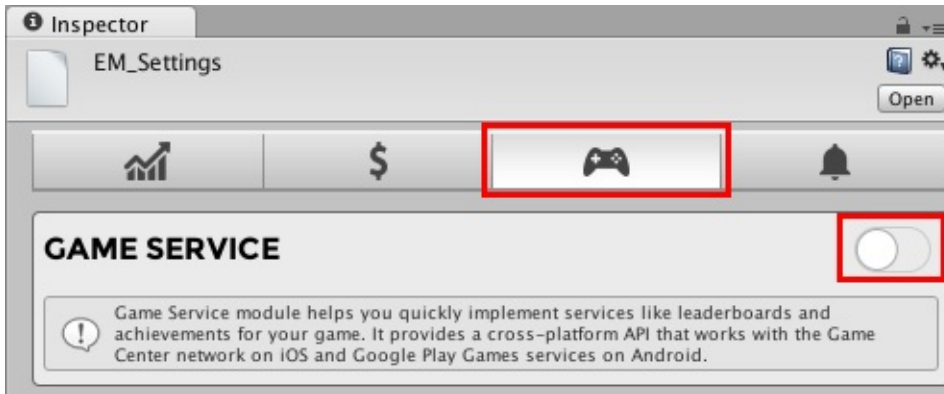
The Game Service module helps you quickly implement leaderboards and achievements for your game. It works with the Game Center network on iOS and Google Play Games services on Android. Here're some highlights of this module:

- **Leverages official plugins**
  - This module is built on top of Unity's GameCenterPlatform on iOS and [Google Play Games plugin](#) on Android
  - GameCenterPlatform is one part of the UnityEngine itself while the other is the official Google Play Games plugin for Unity, so reliability and compatibility can be expected
- **Easy management of leaderboards and achievements**
  - Easy Mobile's custom editor features a friendly interface that help you easily add, edit or remove leaderboards and achievements



# Module Configuration

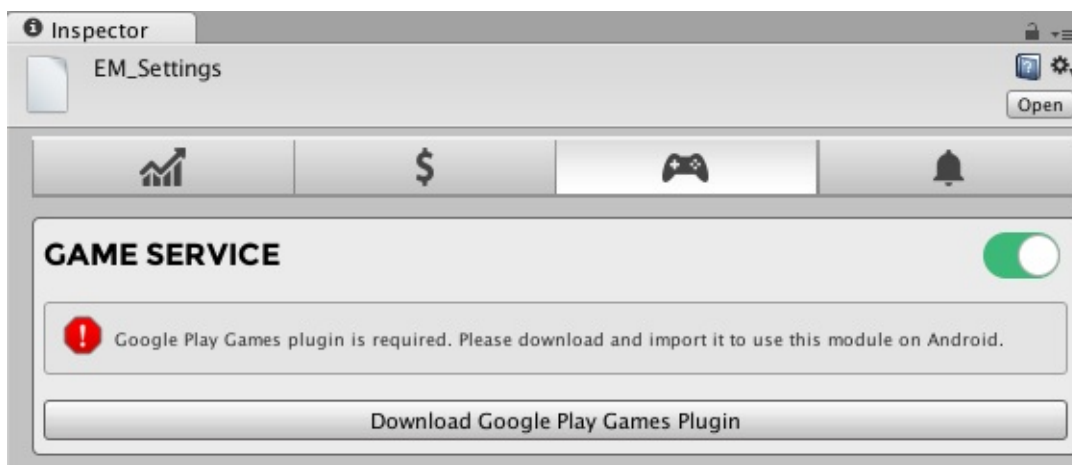
To use the Game Service module you must first enable it. Go to *Window > Easy Mobile > Settings*, select the Game Service tab, then click the right-hand side toggle to enable and start configuring the module.



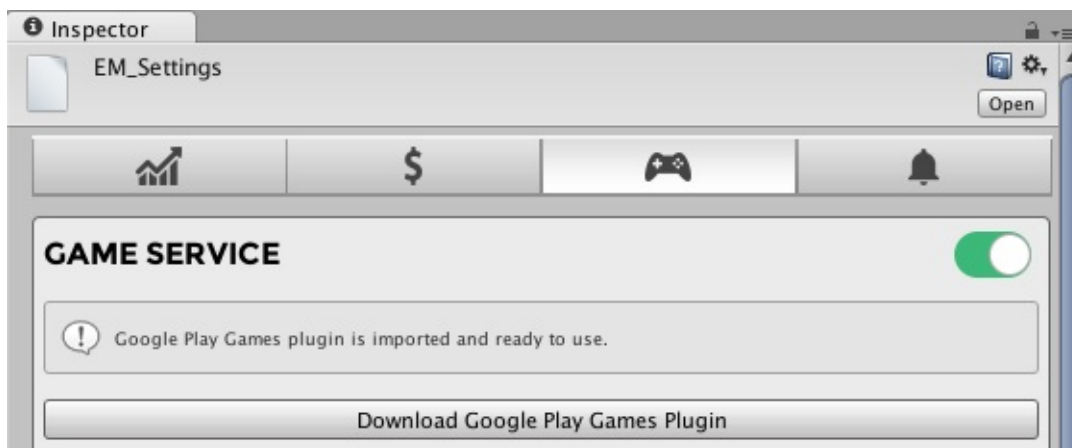
# Android-Specific Setup

## Import Google Play Games plugin for Unity

As stated earlier, this module is built on top of [Google Play Games Plugin](#) on Android. Therefore you need to import it to use the module on this platform. Easy Mobile will automatically detect the availability of the plugin and prompt you to import it if needed. Below is the module settings interface on Android platform when Google Play Games plugin hasn't been imported.

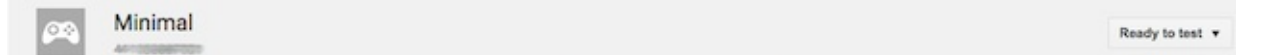


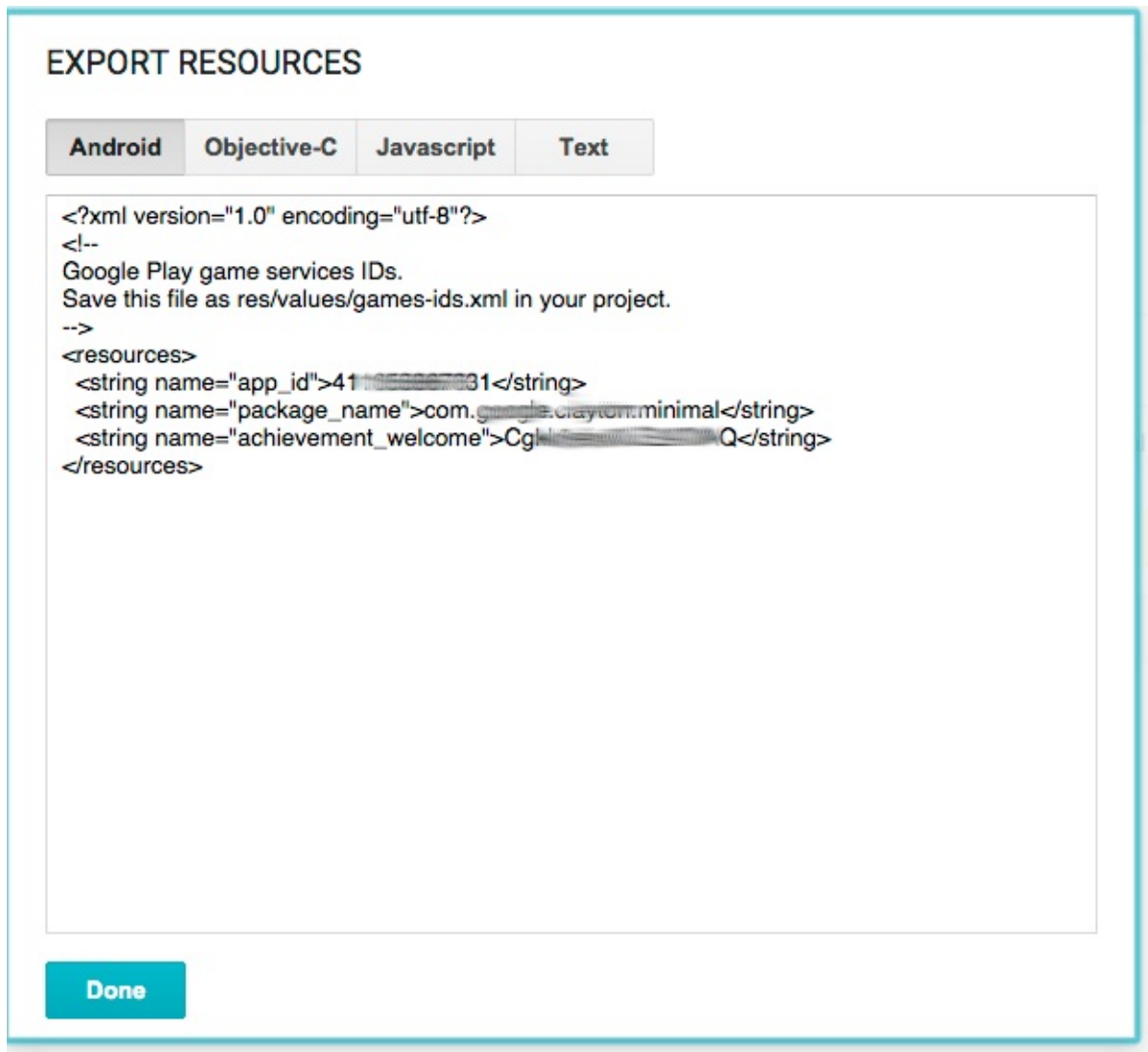
Click the *Download Google Play Games Plugin* button to open the download page, then download the package and import it to your project. Once the import completes the module interface will be updated and ready for you to start with the configuration.



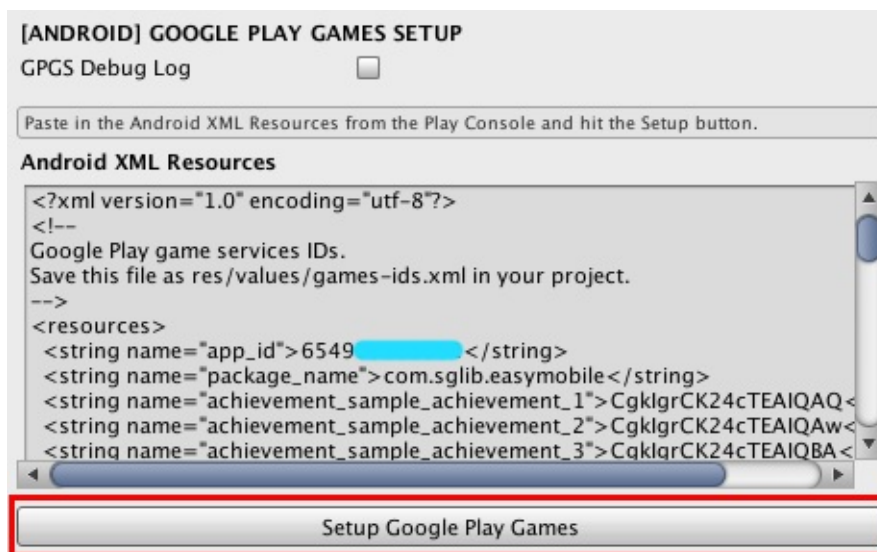
Since we're not using Google Play Games plugin on iOS, the `NO_GPGS` symbol will be defined for iOS platform automatically after the plugin is imported in order to disable it.

## Setup Google Play Games





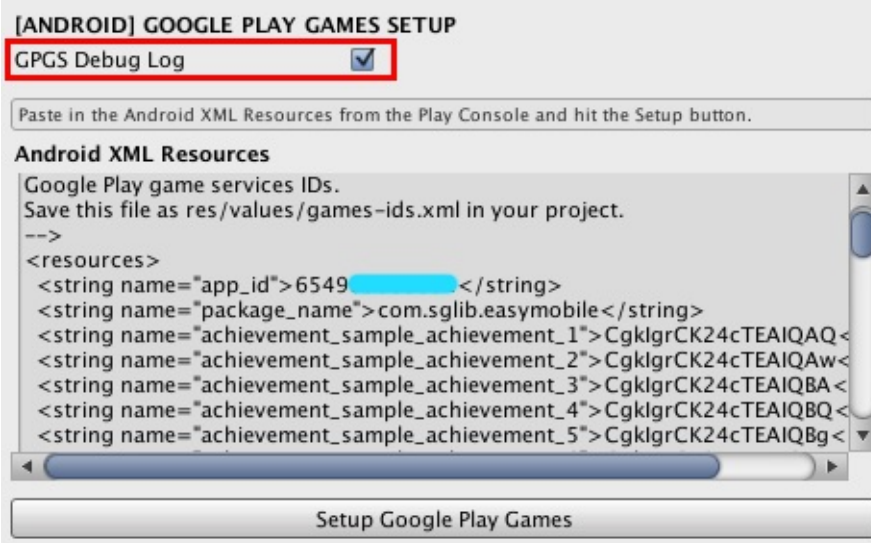
Go back to Unity, in the **[ANDROID] GOOGLE PLAY GAMES SETUP** section, paste the obtained xml resources into the **Android XML Resources** area, then click *Setup Google Play Games*.



After the setup has completed, a new file named `EM_GPGSIds` will be created at `Assets/EasyMobile/Generated`. This file contains the constants of the IDs of all the leaderboards and achievements in your Android game.

## Enable Google Play Games Debug Log

To enable Google Play Games debug log, simply check the *GPGS Debug Log* option in the **[ANDROID] GOOGLE PLAY GAMES SETUP** section.



**[ANDROID] GOOGLE PLAY GAMES SETUP**

**GPGS Debug Log** ☒

Paste in the Android XML Resources from the Play Console and hit the Setup button.

**Android XML Resources**

Google Play game services IDs.  
Save this file as `res/values/games-ids.xml` in your project.

```
-->
<resources>
  <string name="app_id">6549</string>
  <string name="package_name">com.sglib.easymobile</string>
  <string name="achievement_sample_achievement_1">CgklgrCK24cTEAIQAQ<
  <string name="achievement_sample_achievement_2">CgklgrCK24cTEAIQAw<
  <string name="achievement_sample_achievement_3">CgklgrCK24cTEAIQBA<
  <string name="achievement_sample_achievement_4">CgklgrCK24cTEAIQBQ<
  <string name="achievement_sample_achievement_5">CgklgrCK24cTEAIQBg<
```

Setup Google Play Games

# Auto Initialization

Auto initialization is a feature of the Game Service module that initializes the service automatically when the module starts. Initialization is required before any other actions can be done, e.g. reporting scores.

During the initialization, the system will try to authenticate the user by presenting a login popup.

- On iOS, this popup will show up when the app gets focus (brought to foreground) for the first 3 times. If the user refuses to login all these 3 times, the OS will ignore subsequent authentication calls and stop presenting the login popup (to avoid disturbing the user). Otherwise, if the user has logged in successfully, future authentication will take place silently with no login popup presented.
- On Android, we employ a similar approach but you can configure the maximum number of authentication requests before ignoring subsequent ones.

You can configure the auto initialization feature within the **AUTO-INIT CONFIG** section.

| AUTO-INIT CONFIG             |                                     |
|------------------------------|-------------------------------------|
| Auto Init                    | <input checked="" type="checkbox"/> |
| Auto Init Delay              | <input type="text" value="0"/>      |
| [Android] Max Login Requests | <input type="text" value="3"/>      |

- *Auto Init*: uncheck this option to disable the auto initialization feature, you can start the initialization manually from script (see **Scripting** section)
- *Auto Init Delay*: how long after the module start that the initialization should take place
- *[Android] Max Login Requests*: maximum number of authentication requests allowed on Android, before ignoring subsequent ones (in case the user refuses to login)

"Module start" refers to the moment the *Start* method of the module's associated MonoBehaviour (attached to the EasyMobile prefab) runs.

# Leaderboards & Achievements

This section provides a guide to manage leaderboards and managements for your game.

## Before You Begin

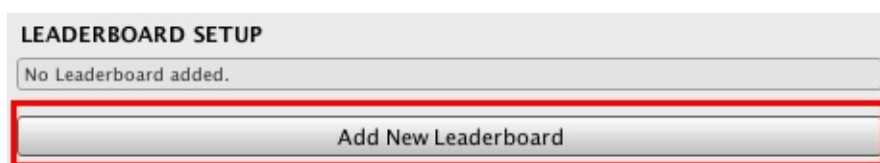
It is assumed that you already configured your game for the targeted gaming networks, i.e. Game Center and Google Play Games. If you're not familiar with the process, here're some useful links:

- Configure for Google Play Games (Android)
  - [Creating a Client ID for you game](#)
  - [Adding leaderboards](#)
  - [Adding achievements](#)
- Configure for Game Center (iOS)
  - [Adding leaderboards and achievements in iTunes Connect](#)

In the **LEADERBOARD SETUP** and **ACHIEVEMENT SETUP** you can add, edit or remove leaderboards and achievements.

## Add a New Leaderboard or Achievement

To add a new leaderboard click the *Add New Leaderboard* button (or *Add New Achievement* button in case of an achievement).



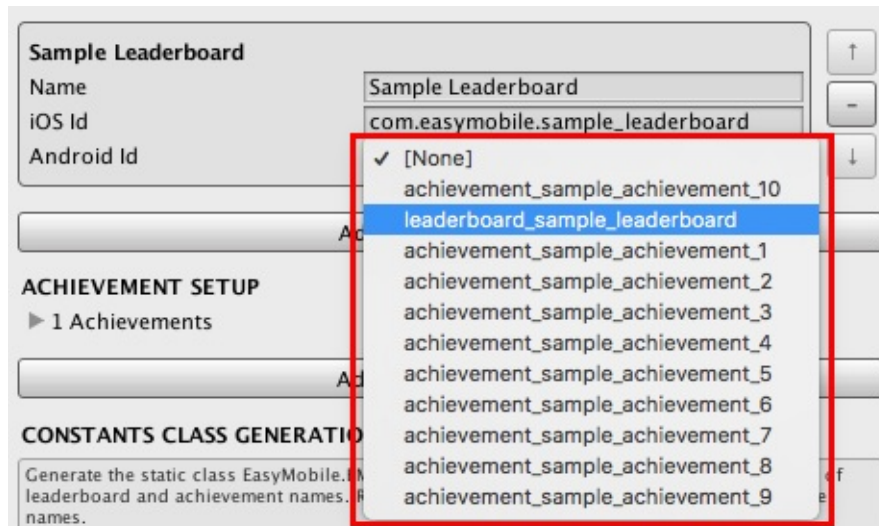
A new empty leaderboard (or achievement) will be added.

Fill in the required information of the leaderboard (or achievement):

- **Name:** the name of this leaderboard (or achievement), this name can be used when reporting scores to this leaderboard (or unlocking this achievement)
- **iOS Id:** the ID of this leaderboard (or achievement) as declared in iTunes Connect

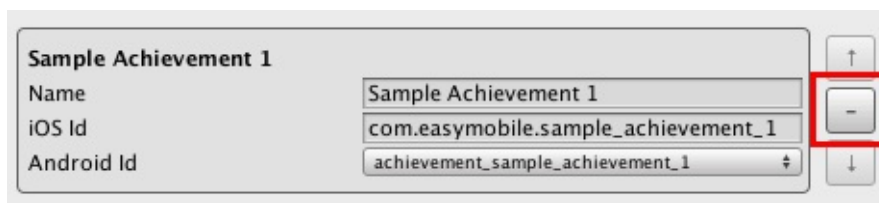
- *Android Id*: the ID of this leaderboard (or achievement) as declared in Google Play Developer Console

Google Play Games' leaderboards and achievements have generated IDs which can be difficult to memorize and cumbersome to copy-and-paste, especially if there are many of them. Thankfully, when you setup Google Play Games, the constants of these IDs are generated automatically (remember that EM\_GPGLIDs file?), allowing Easy Mobile to show a nice dropdown of all defined leaderboard and achievement IDs for you to choose from.



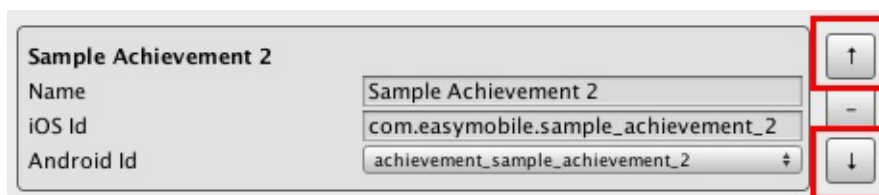
## Remove a Leaderboard or Achievement

To remove a leaderboard (or achievement), simply click the [-] button at the right hand side.



## Arrange Leaderboards or Achievements

You can use the two arrow-up and arrow-down buttons to move a leaderboard (or achievement) upward or downward within its array.



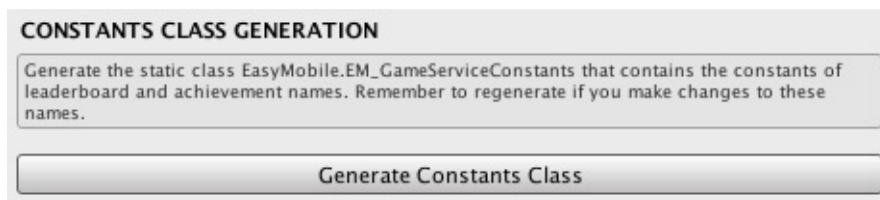




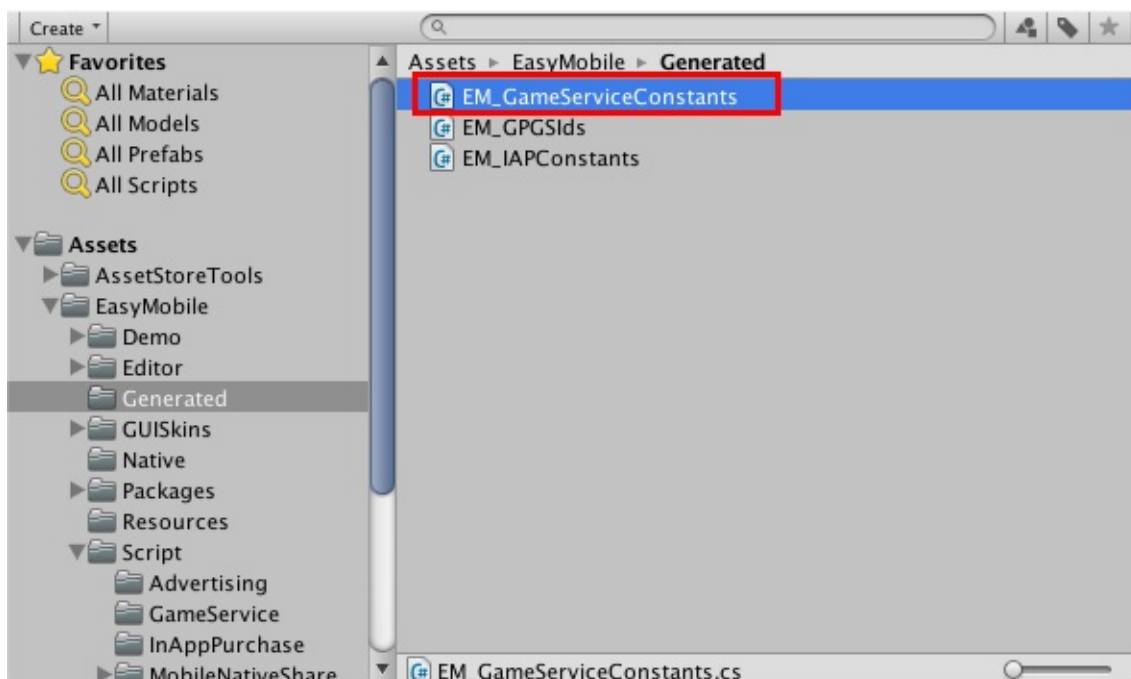
# Game Service Constants Generation

Constants generation is a feature of the Game Service module. It reads the names of all the added leaderboards and achievements and generates a static class named `EM_GameServiceConstants` that contains the constants of these names. Later, you can use these constants when reporting scores to a leaderboard or unlocking an achievement in script instead of typing the names directly, thus help prevent runtime errors due to typos and the likes.

To generate the constants class (you should do this after adding all required leaderboards and achievements), click the *Generate Constants Class* button within the **CONSTANTS CLASS GENERATION** section.



When the process completes, a file named `EM_GameServiceConstants` will be created at *Assets/EasyMobile/Generated*.



# Scripting

This section provides a guide to work with the Game Service API.

You can access all the Game Service API methods via the `GameServiceManager` class under the `EasyMobile` namespace.

## Initialization

Initialization is required before any further API calls can be made. It should only be done once when the app is loaded. If you have enabled the Auto initialization feature, you don't need to initialize in script (see **Auto Initialization** section). Otherwise, if you choose to disable that feature, you can start the initialization in a couple of ways.

- **Managed initialization:** this method respects the *Max Login Requests* value on Android (see **Auto Initialization** section), which means it will ignore all subsequent calls once the user has dismissed the login popup for a number of time determined by *Max Login Requests*
- **Unmanaged initialization:** this method simply initializes the module, on Android it shows the login popup every time as long as the user hasn't been authenticated

On iOS, the system automatically limits the maximum number of login requests to 3 no matter which method is used.

To use the managed initialization method:

```
// Managed init respects the Max Login Requests value
GameServiceManager.ManagedInit();
```

To use the unmanaged initialization method:

```
// Unmanaged init
GameServiceManager.Init();
```

Note that the initialization should be done early and only once, e.g. you can put it in the *Start* method of a *MonoBehaviour*, preferably a singleton one so that it won't run again when the scene reloads.

```
// Initialization in the Start method of a MonoBehaviour script
void Start()
{
    // Managed init respects the Max Login Requests value
    GameServiceManager.ManagedInit();

    // Do other stuff...
}
```

You can check if the module has been initialized and ready for other actions:

```
// Check if initialization has done (the user has been authenticated)
bool isInitialized = GameServiceManager.IsInitialized();
```

## Leaderboards

This section focuses on working with leaderboards.

### Show Leaderboard UI

To show the leaderboard UI (the system view of leaderboards):

```
// Show leaderboard UI
GameServiceManager.ShowLeaderboardUI();
```

You should check if the initialization has finished (the user has been authenticated) before showing the leaderboard UI, and take appropriate actions if the user is not logged in, e.g. show an alert or start another initialization process.

```
// Check for initialization before showing leaderboard UI
if (GameServiceManager.IsInitialized())
{
    GameServiceManager.ShowLeaderboardUI();
}
else
{
    #if UNITY_ANDROID
    GameServiceManager.Init();    // start a new initialization process
    #elif UNITY_IOS
    Debug.Log("Cannot show leaderboard UI: The user is not logged in to Game Center.");
    ;
    #endif
}
```

### Report Scores

To report scores to a leaderboard you need to specify the name of that leaderboard.

It is strongly recommended that you use the constants of leaderboard names in the generated `EM_GameServiceConstants` class (see **Game Service Constants Generation** section) instead of typing the names directly in order to prevent runtime errors due to typos and the likes.

```
// Report a score of 100
// EM_GameServiceConstants.Sample_Leaderboard is the generated name constant
// of a leaderboard named "Sample Leaderboard"
GameServiceManager.ReportScore(100, EM_GameServiceConstants.Sample_Leaderboard);
```

## Load Local User's Score

You can load the score of the local user (the authenticated user) on a leaderboard, to do so you need to specify the name of the leaderboard to load score from and a callback to be called when the score is loaded.

```
// Load the local user's score from the specified leaderboard
// EM_GameServiceConstants.Sample_Leaderboard is the generated name constant
// of a leaderboard named "Sample Leaderboard"
GameServiceManager.LoadLocalUserScore(EM_GameServiceConstants.Sample_Leaderboard, OnLocalUserScoreLoaded);

// Score loaded callback
void OnLocalUserScoreLoaded(string leaderboardName, IScore score)
{
    if (score != null)
    {
        Debug.Log("Your score is: " + score.value);
    }
    else
    {
        Debug.Log("You don't have any score reported to leaderboard " + leaderboardName);
    }
}
```

## Load Scores

You can load a set of scores from a leaderboard with which you can specify the start position to load score, the number of scores to load, as well as the time scope and user scope.

```
// Load a set of 20 scores starting from rank 10 in Today time scope and Global user s
cope
// EM_GameServiceConstants.Sample_Leaderboard is the generated name constant
// of a leaderboard named "Sample Leaderboard"
GameServiceManager.LoadScores(
    EM_GameServiceConstants.Sample_Leaderboard,
    10,
    20,
    TimeScope.Today,
    UserScope.Global,
    OnScoresLoaded
);

// Scores loaded callback
void OnScoresLoaded(string leaderboardName, IScore[] scores)
{
    if (scores != null && scores.Length > 0)
    {
        Debug.Log("Loaded " + scores.Length + " from leadeboard " + leaderboardName);
        foreach (IScore score in scores)
        {
            Debug.Log("Score: " + score.value + "; rank: " + score.rank);
        }
    }
    else
    {
        Debug.Log("No score loaded.");
    }
}
```

You can also load the default set of scores, which contains 25 scores around the local user's score in the *AllTime* time scope and *Global* user scope.

```
// Load the default set of scores
// EM_GameServiceConstants.Sample_Leaderboard is the generated name constant
// of a leaderboard named "Sample Leaderboard"
GameServiceManager.LoadScores(EM_GameServiceConstants.Sample_Leaderboard, OnScoresLoaded);

// Scores loaded callback
void OnScoresLoaded(string leaderboardName, IScore[] scores)
{
    if (scores != null && scores.Length > 0)
    {
        Debug.Log("Loaded " + scores.Length + " from leadeboard " + leaderboardName);
        foreach (IScore score in scores)
        {
            Debug.Log("Score: " + score.value + "; rank: " + score.rank);
        }
    }
    else
    {
        Debug.Log("No score loaded.");
    }
}
```

## Get All Leaderboards

You can obtain an array of all leaderboards created in the module settings interface:

```
// Get the array of all leaderboards created in the Game Service module settings
// Leaderboard is the class representing a leaderboard as declared in the module settings
// The GameService property of EM_Settings class holds the settings of this module
Leaderboard[] leaderboards = EM_Settings.GameService.Leaderboards;

// Print all leaderboard names
foreach (Leaderboard ldb in leaderboards)
{
    Debug.Log("Leaderboard name: " + ldb.Name);
}
```

## Achievements

This section focuses on working with achievements.

### Show Achievement UI

To show the achievements UI (the system view of achievements):

```
// Show achievements UI
GameServiceManager.ShowAchievementsUI();
```

You should check if the initialization has finished (the user has been authenticated) before showing the achievements UI, and take appropriate actions if the user is not logged in, e.g. show an alert or start another initialization process.

```
// Check for initialization before showing achievements UI
if (GameServiceManager.IsInitialized())
{
    GameServiceManager.ShowAchievementsUI();
}
else
{
    #if UNITY_ANDROID
    GameServiceManager.Init();    // start a new initialization process
    #elif UNITY_IOS
    Debug.Log("Cannot show achievements UI: The user is not logged in to Game Center."
);
    #endif
}
```

## Reveal an Achievement

To reveal a hidden achievement, simply specify its name.

As in the case of leaderboards, it is strongly recommended that you use the constants of achievement names in the generated `EM_GameServiceConstants` class instead of typing the names directly.

```
// Reveal a hidden achievement
// EM_GameServiceConstants.Sample_Achievement is the generated name constant
// of an achievement named "Sample Achievement"
GameServiceManager.RevealAchievement(EM_GameServiceConstants.Sample_Achievement);
```

## Unlock an Achievement

To unlock an achievement:

```
// Unlock an achievement
// EM_GameServiceConstants.Sample_Achievement is the generated name constant
// of an achievement named "Sample Achievement"
GameServiceManager.UnlockAchievement(EM_GameServiceConstants.Sample_Achievement);
```

## Report Incremental Achievement's Progress



To report the progress of an incremental achievement:

```
// Report a progress of 50% for an incremental achievement
// EM_GameServiceConstants.Sample_Incremental_Achievement is the generated name constant
// of an incremental achievement named "Sample Incremental Achievement"
GameServiceManager.UnlockAchievement(EM_GameServiceConstants.Sample_Incremental_Achievement, 50.0f);
```

## Get All Achievements

You can obtain an array of all achievements created in the module settings interface:

```
// Get the array of all achievements created in the Game Service module settings
// Achievement is the class representing an achievement as declared in the module settings
// The GameService property of EM_Settings class holds the settings of this module
Achievement[] achievements = EM_Settings.GameService.Achievements;

// Print all achievement names
foreach (Achievement acm in achievements)
{
    Debug.Log("Achievement name: " + acm.Name);
}
```

## Load User Profiles

You can load the profiles of friends of the local (authenticated) user. When the loading completes the provided callback will be invoked.

```
// Load the local user's friend list
GameServiceManager.LoadFriends(OnFriendsLoaded);

// Friends loaded callback
void OnFriendsLoaded(IUserProfile[] friends)
{
    if (friends.Length > 0)
    {
        foreach (IUserProfile user in friends)
        {
            Debug.Log("Friend's name: " + user.userName + "; ID: " + user.id);
        }
    }
    else
    {
        Debug.Log("Couldn't find any friend.");
    }
}
```

You can also load user profiles by providing their IDs.

```
// Load the profiles of the users with provided IDs
// idArray is the (string) array of the IDs of the users to load profiles
GameServiceManager.LoadUsers(idArray, OnUsersLoaded);

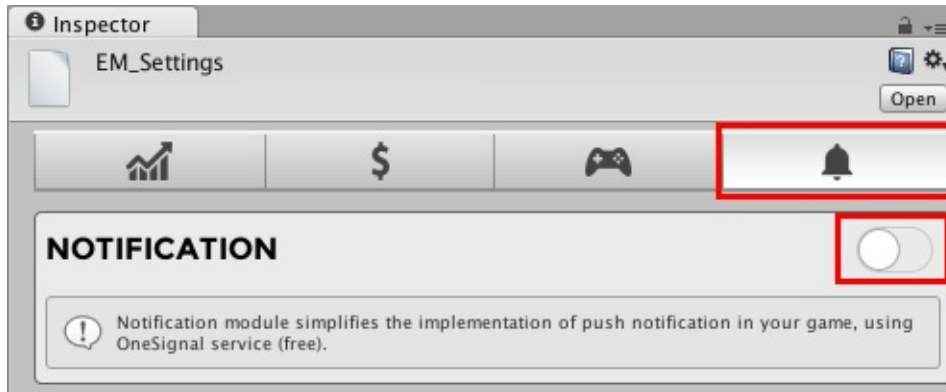
// Users loaded callback
void OnUsersLoaded(IUserProfile[] users)
{
    if (users.Length > 0)
    {
        foreach (IUserProfile user in users)
        {
            Debug.Log("User's name: " + user.userName + "; ID: " + user.id);
        }
    }
    else
    {
        Debug.Log("Couldn't find any user with the specified IDs.");
    }
}
```

# Notification

The Notification module helps you quickly setup you game for receiving push notifications. It is compatible with [OneSignal](#), a free, popular cross-platform push notification delivery service.

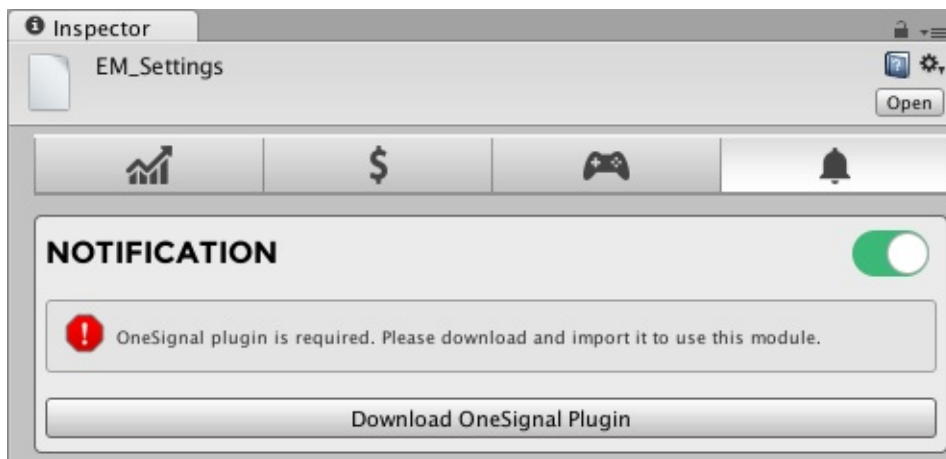
# Module Configuration

To use the Notification module you must first enable it. Go to *Window > Easy Mobile > Settings*, select the Notification tab, then click the right-hand side toggle to enable and start configuring the module.

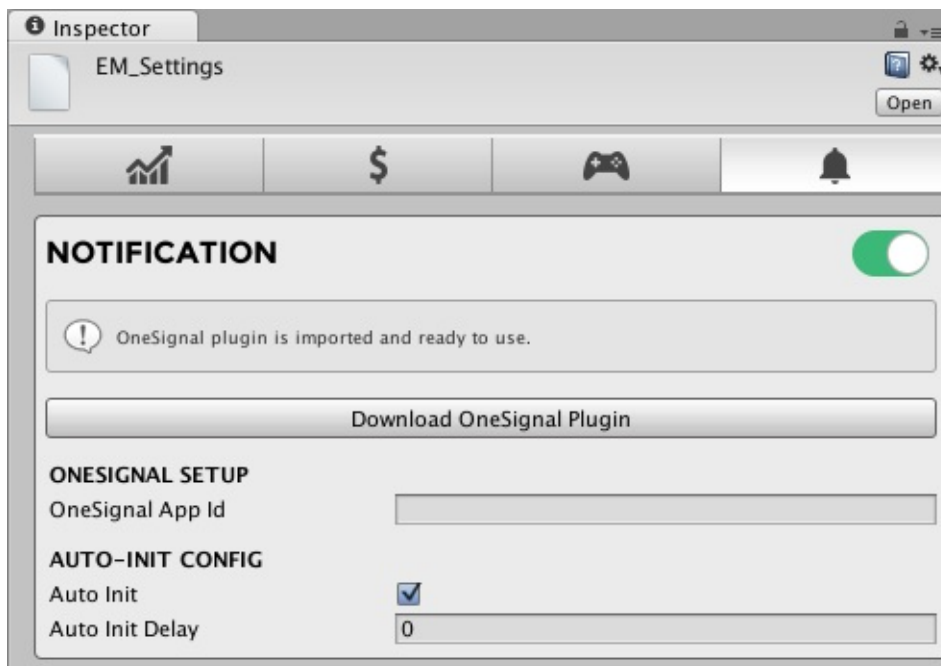


## Import OneSignal Plugin

Using OneSignal service requires [OneSignal plugin for Unity](#). Easy Mobile will automatically check for the availability of the plugin and prompt you to import it if needed. Below is the module settings interface when OneSignal plugin hasn't been imported.



Click the *Download OneSignal Plugin* button to open the download page, then download the package and import it to your project. Once the import completes the settings interface will be updated and ready for you to start configuring.

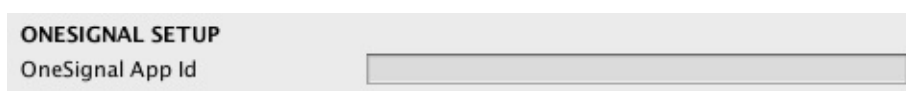


## Setup OneSignal

### Before You Begin

Before setting up OneSignal in Unity, you must first generate appropriate credentials for your targeted platforms. If you're not familiar with the process, please follow the guides listed [here](#). You should also follow the instructions included in that document on performing necessary setup when building for each platform.

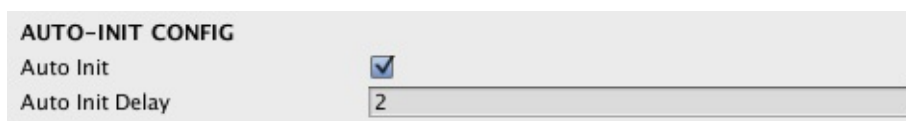
In the **ONESIGNAL SETUP** section, enter your OneSignal App ID.



## Auto Initialization

Auto initialization is a feature of the Notification module that initializes the service automatically when the module starts. You can configure this feature in the **AUTO-INIT CONFIG** section.

On iOS, a popup will appear during the first initialization following the app install to ask for the user's permission to receive push notifications for your game.



- *Auto Init*: uncheck this option to disable the auto initialization feature, you can start the initialization manually from script (see **Scripting** section)

- *Auto Init Delay*: how long after the module start that the initialization should take place

"Module start" refers to the moment the *Start* method of the module's associated MonoBehavior (attached to the EasyMobile prefab) runs.

# Scripting

This section provides a guide to work with the Notification API.

You can access all the Notification API methods via the `NotificationManager` class under the `EasyMobile` namespace.

## Initialization

Before receiving push notifications, the service needs to be initialized. This initialization should only be taken once when the app is loaded, and before any other calls to the API are made. If you have enabled the Auto initialization feature (see **Module Configuration** section), you don't need to start the initialization from script. Otherwise, if you choose to disable that feature, you can initialize the service using the *Init* method.

```
// Initialize push notification service
NotificationManager.Init();
```

Note that the initialization should be done early and only once, e.g. you can put it in the *Start* method of a `MonoBehaviour`, preferably a singleton one so that it won't run again when the scene reloads.

```
// Initialization in the Start method of a MonoBehaviour script
void Start()
{
    // Initialize push notification service
    NotificationManager.Init();

    // Do other stuff...
}
```

## The *NotificationOpened* Event

A *NotificationOpened* event will be fired whenever a push notification is opened and your app is put to foreground. You can listen to this event and take appropriate actions, e.g. take the user to the store page of your game to download an update when it's available.

You should subscribe to this event as early as possible, preferably as soon as your app is loaded, e.g. in the *OnEnable* method of a `MonoBehaviour` script in your first scene.

```
// Subscribe to the event
void OnEnable()
{
    NotificationManager.NotificationOpened += OnNotificationOpened;
}

// The event handler
void OnNotificationOpened(string message, string actionID, Dictionary<string, object>
additionalData, bool isAppInFocus)
{
    Debug.Log("Push notification received!");
    Debug.Log("Message: " + message);

    if (additionalData != null)
    {
        // Check if a new update is available, suppose we use
        // a key called "newUpdate" to signal the availability of one
        if (additionalData.ContainsKey("newUpdate"))
        {
            // Here you should ask the users if they want to update
            // and open the download page if they do...
        }
    }
}

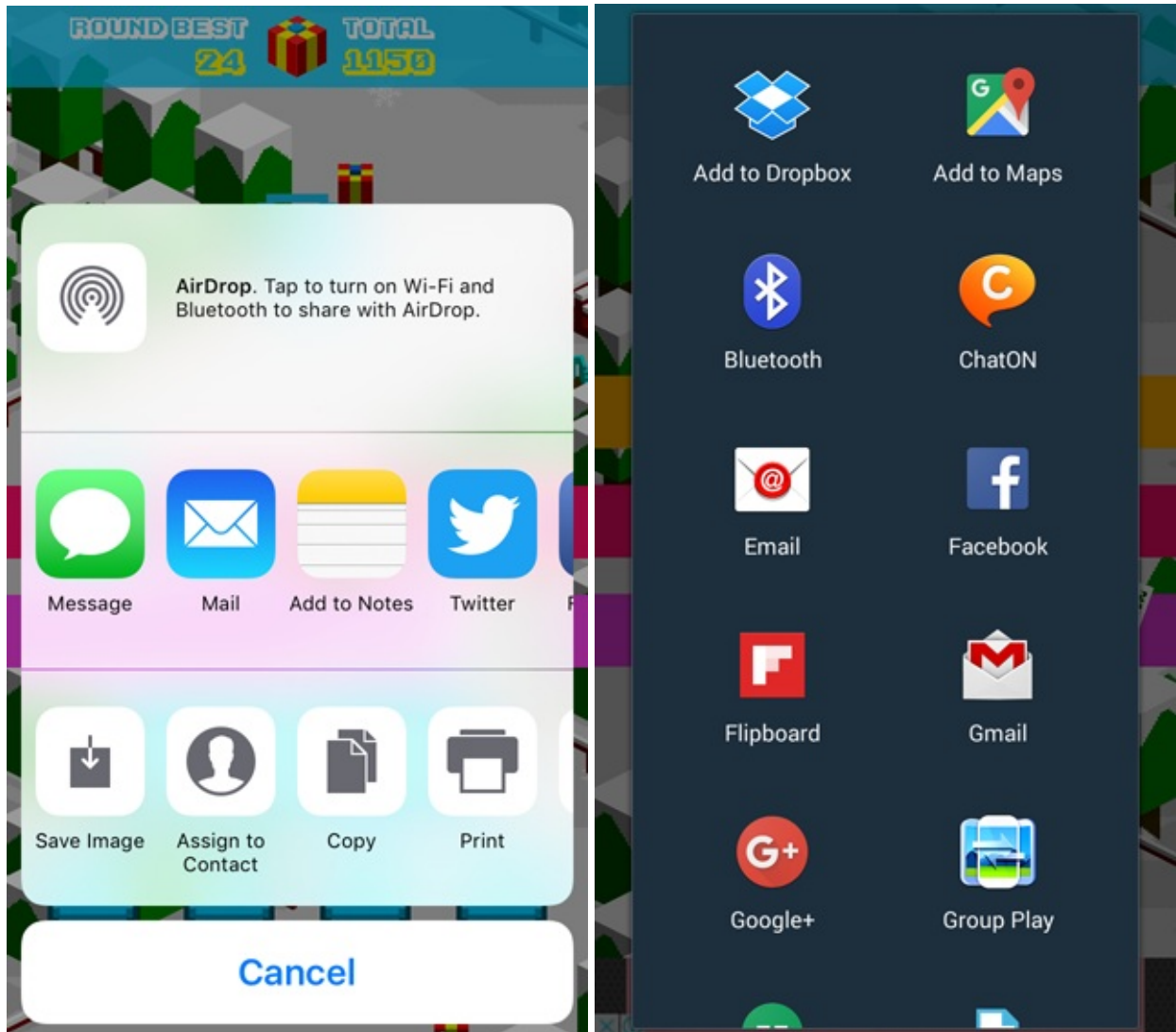
// Unsubscribe
void OnDisable()
{
    NotificationManager.NotificationOpened -= OnNotificationOpened;
}
```



## Native Sharing

The Native Sharing module helps you easily share texts and images to social networks including Facebook, Twitter and Google+ using the native sharing functionality. In addition, it also provides convenient methods to capture the screenshots to be shared.

Below are the sharing interfaces on iOS and Android, respectively.



## Enable External Write Permission on Android

For this module to function on Android, it is necessary to enable the permission to write to external storage. To do so, go to *Edit > Project Settings > Player*, select *Android settings* tab, then locate the **Configuration** section and set the *Write Permission* to *External (SDCard)*.

Configuration

Scripting Backend

Mono2x

Mute Other Audio Sources\*

☐

Disable HW Statistics

☐

Device Filter

FAT (ARMv7+x86)

Install Location

Prefer External

Internet Access

Auto

Write Permission

External (SDCard)

Android TV Compatibility

☒

Android Game

☒

Android Gamepad Support Level

Works with D-pad

Scripting Define Symbols

EASY\_MOBILE

# Scripting

This section provides a guide to work with Native Sharing API.

You can access all the Native Sharing API methods via the `MobileNativeShare` class under the `EasyMobile` namespace.

## Capture Screenshots

To capture the device's screenshot, you have a few options.

### Capture and Save a Screenshot as PNG Image

To capture and save a screenshot of the whole device screen, simply specify the file name to be saved. This screenshot will be saved as a PNG image in the directory pointed by [Application.persistentDataPath](#). Note that this method, as well as other screenshot capturing methods, needs to be called at the end of a frame (when the rendering has done) for it to produce a proper image. Therefore you should call it within a coroutine after `WaitForEndOfFrame()`.

```
// Coroutine that captures and saves a screenshot
IEnumerator SaveScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // The SaveScreenshot() method returns the path of the saved image
    // The provided file name will be added a ".png" extension automatically
    string path = MobileNativeShare.SaveScreenshot("screenshot");
}
```

You can also captures and saves just a portion of the screen:

```
// Coroutine that captures and saves a portion of the screen
IEnumerator SaveScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // Capture the portion of the screen starting at (50, 50),
    // has a width of 200 and a height of 400 pixels.
    string path = MobileNativeShare.SaveScreenshot(50, 50, 200, 400, "screenshot");
}
```

## Capture a Screenshot into a Texture2D

In some cases you may want to capture a screenshot and obtain a Texture2D object of it instead of saving to disk, e.g. to create a sprite from the texture and display it in-game.

```
// Coroutine that captures a screenshot and generates a Texture2D object of it
IEnumerator CaptureScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // Create a Texture2D object of the screenshot using the CaptureScreenshot() method
    Texture2D texture = MobileNativeShare.CaptureScreenshot();
}
```

Similar to the case above, you can also capture only a portion of the screen.

```
// Coroutine that captures a portion of the screenshot and generates a Texture2D object of it
IEnumerator CaptureScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

    // Create a Texture2D object of the screenshot using the CaptureScreenshot() method
    // The captured portion starts at (50, 50) and has a width of 200, a height of 400 pixels.
    Texture2D texture = MobileNativeShare.CaptureScreenshot(50, 50, 200, 400);
}
```

Note that screenshot capturing should be done at the end of the frame.

## Sharing

To share an image you also have a few options. You can also attach a message to be shared with the image.

Due to Facebook policy, pre-filled messages will be ignored when sharing to this network, i.e. sharing messages must be written by the user.

### Share a Saved Image

You can share a saved image by specifying its path.

```
// Share a saved image
// Suppose we have a "screenshot.png" image stored in the persistentDataPath,
// we'll construct its path first
string path = Path.Combine(Application.persistentDataPath, "screenshot.png");

// Share the image with the path, a sample message and an empty subject
MobileNativeShare.ShareImage(path, "This is a sample message", "");
```

## Share a Texture2D

You can also share a Texture2D object obtained some point before the sharing time. Internally, this method will also create a PNG image from the Texture2D, save it to the persistentDataPath, and finally share that image.

```
// Share a Texture2D
// sampleTexture is a Texture2D object captured some time before
// This method saves the texture as a PNG image named "screenshot.png" in persistentDataPath,
// then shares it with a sample message and an empty subject
MobileNativeShare.ShareTexture2D(sampleTexture, "screenshot", "This is a sample message", "");
```

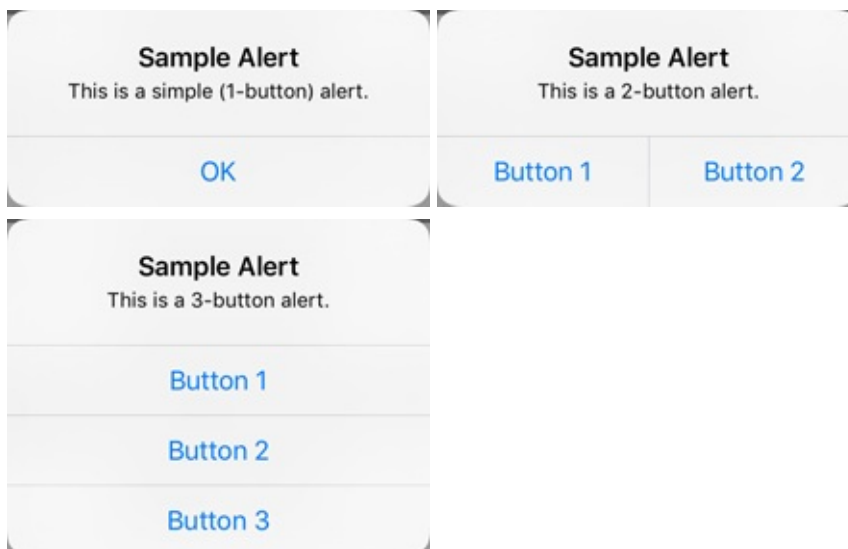
# Native UI

The Native UI module allows you to access native mobile UI elements such as alerts and dialogs. This module requires no configuration and all tasks can be done from script.

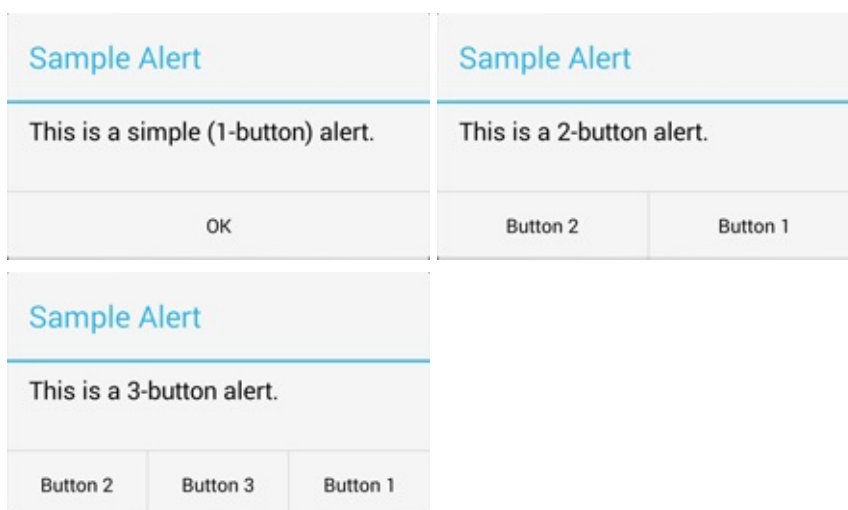
## Alerts

Alerts are useful in providing the users contextual information, asking for confirmation or prompting them to make a selection out of several options. An alert can have one, two or three buttons with it.

Below are the three types of alert on iOS.



And below are the three types of alert on Android.



## Toasts

Toasts are short messages displayed at the bottom of the screen. They automatically disappear after a timeout. Toasts are available on Android only. Below is a sample toast message.



Developed by Sule

# Scripting

This section provides a guide to work with Native UI API.

You can access all the Native UI API methods via the `MobileNativeUI` class under the `EasyMobile` namespace.

## Alerts

Alerts are available on both iOS and Android platform and can have up to three buttons.

Only one alert can be shown at a time.

Simple (one-button) alerts are useful in giving the user contextual information. To show a simple alert with the default OK button, you only need to provide a title and a message for the alert:

```
// Show a simple alert with OK button
MobileNativeAlert alert = MobileNativeUI.Alert("Sample Alert", "This is a sample alert
with an OK button.");
```

You can also show a one-button alert with a custom button label.

```
// Show an alert with a button labeled as "Got it"
MobileNativeAlert alert = MobileNativeUI.Alert(
    "Sample Alert",
    "This is a sample alert with a custom button.",
    "Got it"
);
```

Two-button alerts can be useful when needing to ask for user confirmation. To show a two-button alert, you need to specify the labels of these two buttons.

```
// Show a two-button alert with the buttons labeled as "Button 1" & "Button 2"
MobileNativeAlert alert = MobileNativeUI.ShowTwoButtonAlert(
    "Sample Alert",
    "This is a two-button alert.",
    "Button 1",
    "Button 2"
);
```



Three-button alerts can be used to present the user with several options, a typical usage of it is to implement the Rate Us popup. To show a three-button alert, you need to specify the labels of the three buttons.

```
// Show a three-button alert with the buttons labeled as "Button 1", "Button 2" & "Button 3"
MobileNativeAlert alert = MobileNativeUI.ShowThreeButtonAlert(
    "Sample Alert",
    "This is a three-button alert.",
    "Button 1",
    "Button 2",
    "Button 3"
);
```

Whenever an alert is shown, a *MobileNativeAlert* object is returned, when the alert is closed, this object will fire an *OnComplete* event and then destroy itself. The argument of this event is the index of the clicked button. You should listen to this event and take appropriate action depending on the button selected.

```
// Show a three button alert and handle its OnComplete event
MobileNativeAlert alert = MobileNativeUI.ShowThreeButtonAlert(
    "Sample Alert",
    "This is a three-button alert.",
    "Button 1",
    "Button 2",
    "Button 3"
);

// Subscribe to the event
if (alert != null)
{
    alert.OnComplete += OnAlertCompleteHandler;
}

// The event handler
void OnAlertCompleteHandler(int buttonIndex)
{
    switch (buttonIndex)
    {
        case 0:
            // Button 1 was clicked
            break;
        case 1:
            // Button 2 was clicked
            break;
        case 2:
            // Button 3 was clicked
            break;
        default:
            break;
    }
}
```

## Toasts

Toast is a short message displayed at the bottom of the screen and automatically disappears after a timeout. Toasts are available only on Android platform. To show a toast message:

```
// Show a sample toast message
MobileNativeUI.ShowToast("This is a sample Android toast");
```

# Release Notes

## Version 1.0.0

First release.