

# 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
AdColony	1.2.1.1.1
AdMob	1.2.1.1.2
Chartboost	1.2.1.1.3
Heyzap	1.2.1.1.4
Unity Ads	1.2.1.1.5
Automatic Ad Loading	1.2.1.2
Default Ad Networks	1.2.1.3
Scripting	1.2.2
Game Service	1.3
Module Configuration	1.3.1
Android-Specific Setup	1.3.1.1
Auto Initialization	1.3.1.2
Leaderboards & Achievements	1.3.1.3
Game Service Constants Generation	1.3.1.4
Scripting	1.3.2
GIF	1.4
Setup	1.4.1
Scripting	1.4.2
In-App Purchasing	1.5
Module Configuration	1.5.1
Enable Unity IAP	1.5.1.1
Target Android Store	1.5.1.2
Receipt Validation	1.5.1.3

---

Product Management	1.5.1.4
IAP Constants Generation	1.5.1.5
Scripting	1.5.2
Native Sharing	1.6
Scripting	1.6.1
Native UI	1.7
Scripting	1.7.1
Notification	1.8
Module Configuration	1.8.1
Scripting	1.8.2
Utilities	1.9
Rating Request	1.9.1
Configuration	1.9.1.1
Scripting	1.9.1.2
Release Notes	1.10
Upgrade Guide	1.11

---

# 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 the following modules:

- **Advertising**

- Compatible with AdColony, 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

- **Game Service**

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

- **GIF**

- Records screen, plays recorded clips and exports GIF images
- High-performance, mobile-friendly GIF encoder
- Giphy upload API for sharing GIF to social networks

- **In-App Purchasing**

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

- **Native Sharing**

- Shares texts, URLs and images using the native sharing functionality

- **Native UI**

- Alerts and dialogs
- Toasts (Android only)

- **Notification**

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

- **Utilities**

- Rating request: an effective way to ask for rating using the system-provided rating prompt of iOS (10.3+) and a native, highly customizable popup on Android

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.

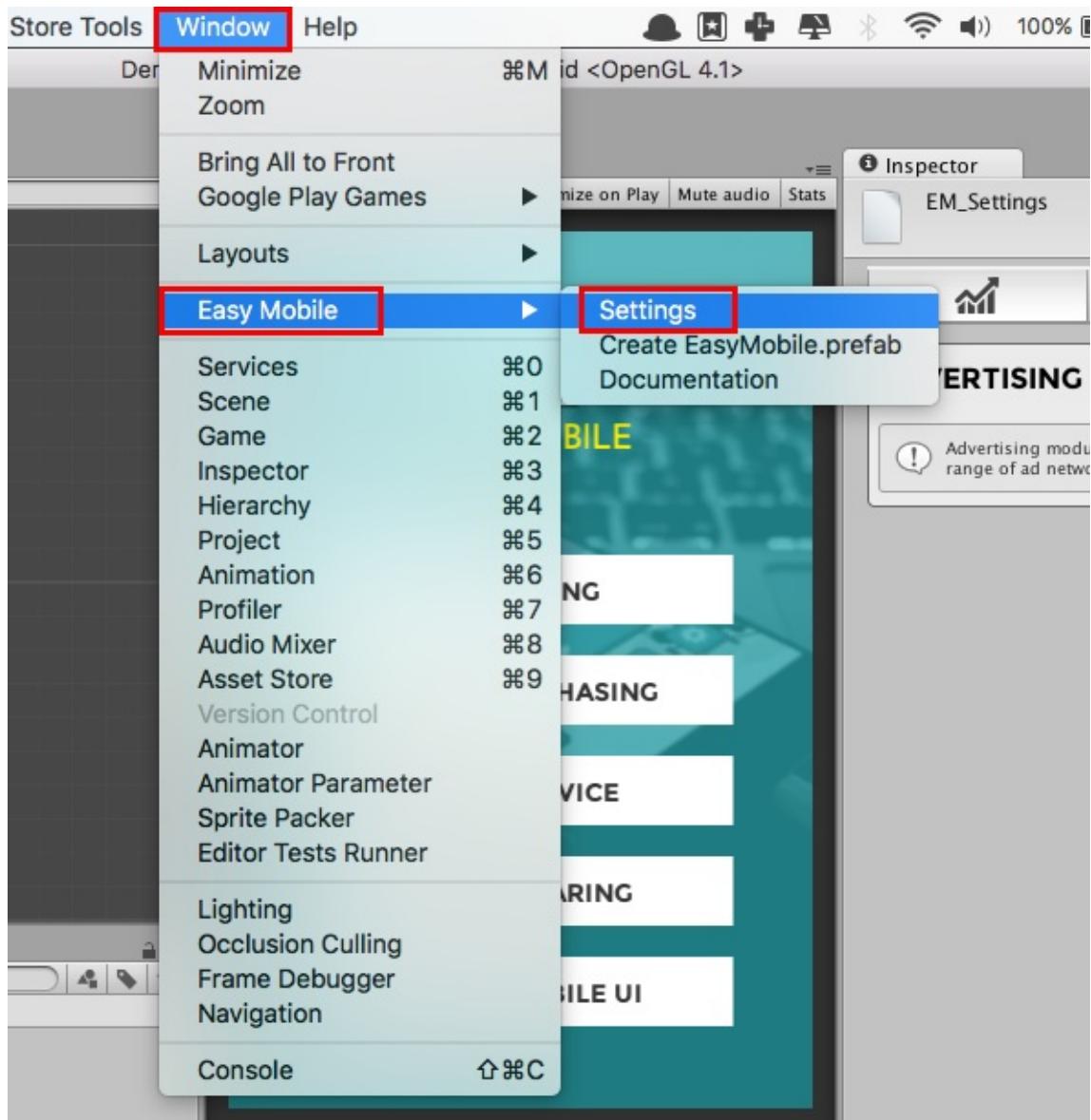
# Using Easy Mobile

Using Easy Mobile involves 3 steps:

- 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

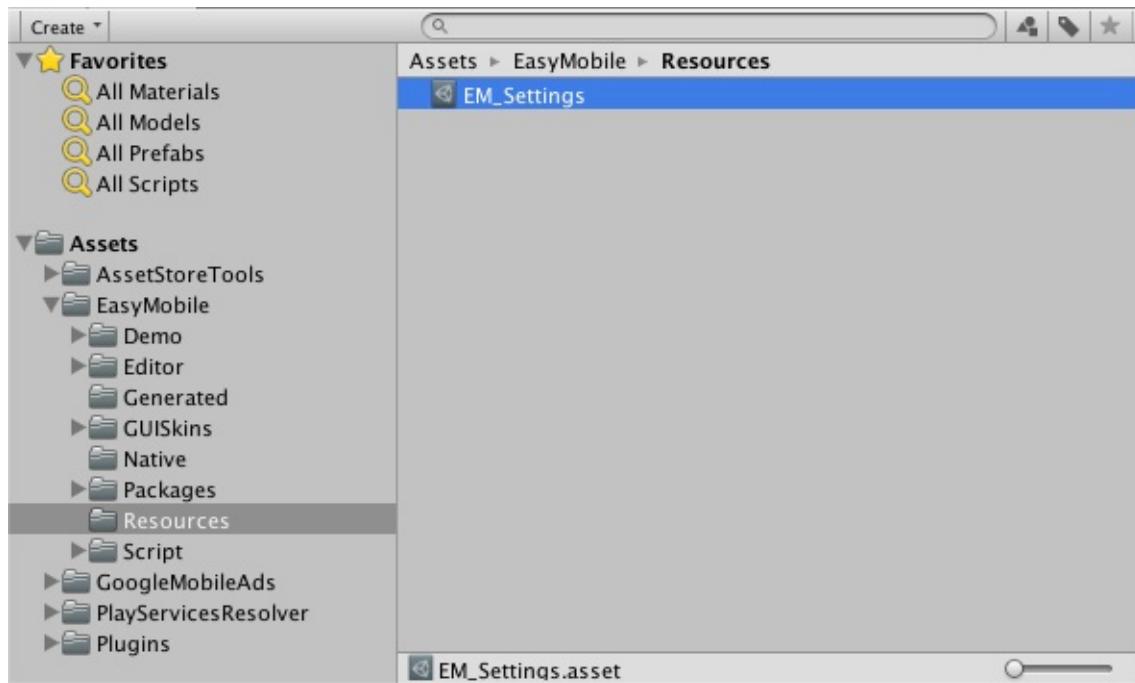
## Configuration

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.

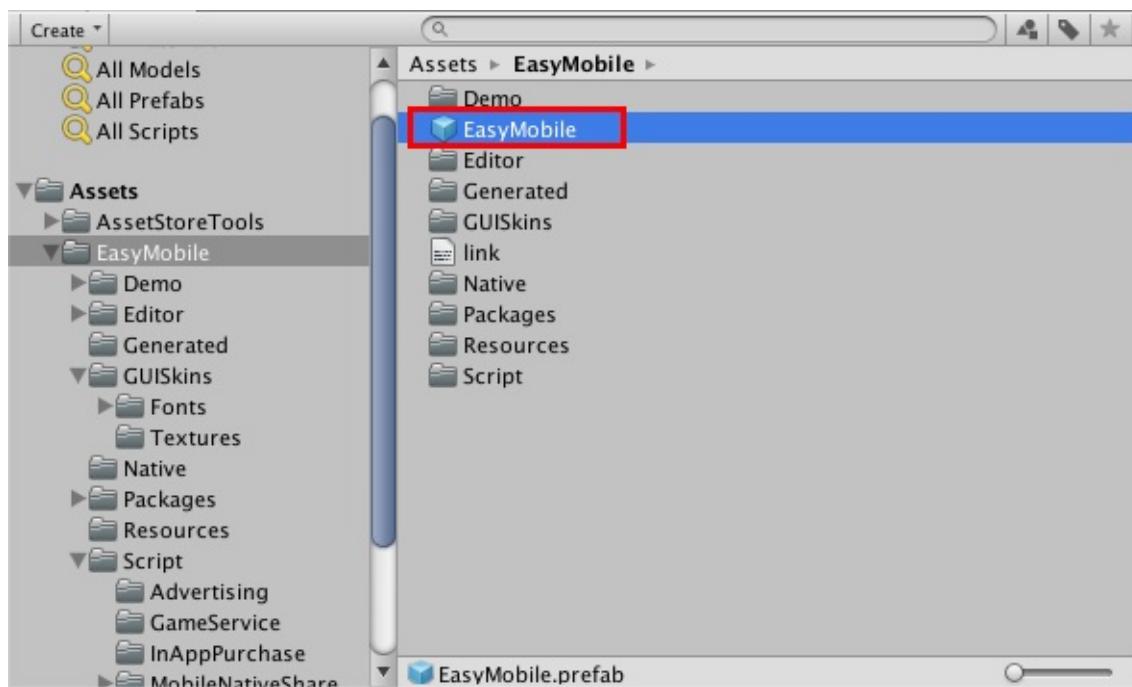
All these settings are stored in the `EM_Settings` object, which is a `ScriptableObject` created automatically after importing the plugin and is located at `Assets/EasyMobile/Resources`. You can also access this `EM_Settings` class from script and via its properties accessing each module settings in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



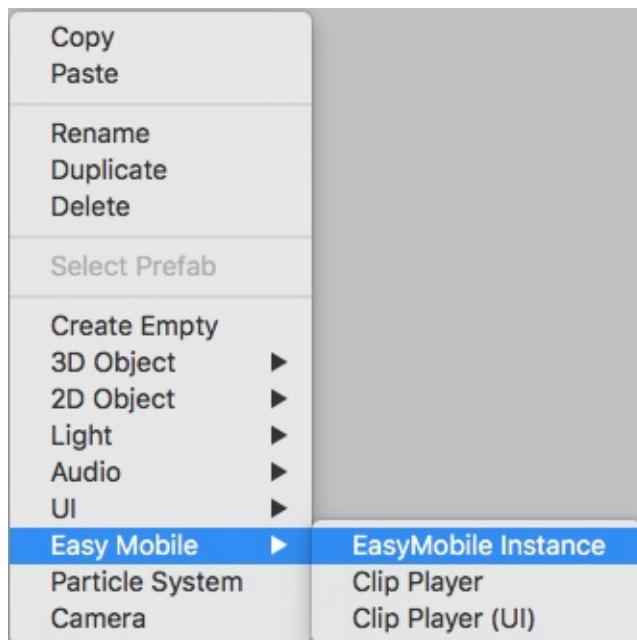
## EasyMobile Prefab

For the plugin to function properly it is required that an instance of 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.

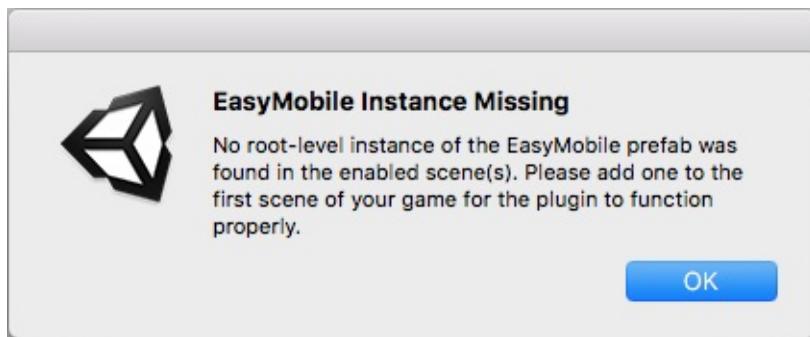
It is advisable to add the EasyMobile prefab to the first scene in your game so that the modules have time to initialize before you actually use them. Likewise, this will allow the automatic ad loading process to start soon and the ads will be more likely available when needed.



To add an EasyMobile instance to your scene, simply right-click in the Hierarchy window to open the context menu (as you would when creating Unity built-in objects), then select *Easy Mobile > EasyMobile Instance*. Alternatively, you can just drag the prefab to the Hierarchy window, only make sure to make it a root-level object (parentless).



If you're using Unity 5.6 or newer, you'll get a warning if you start an iOS or Android build without having added the EasyMobile instance to one of your scenes.



## Scripting

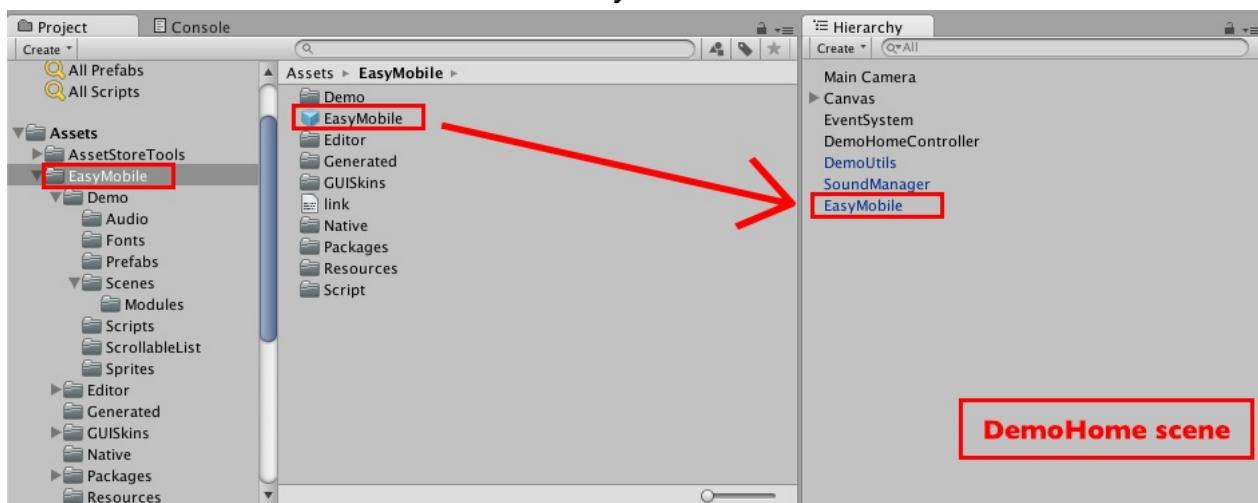
Easy Mobile API is written in C# and is put under the namespace `EasyMobile`. Therefore, you need to add the following statement to the top of your script in order to access its API methods.

```
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 an instance of the `EasyMobile` prefab to the `DemoHome` scene located in the `Assets/EasyMobile/Demo/Scenes` folder.



## Debug.Log Stop Working?

Once you added EasyMobile prefab to your scene, it will automatically configure the default logger such that *it is only enabled in the editor or in development builds*. Therefore, unless you're running the game in the editor or creating a development build, all the Debug.Log calls will be suppressed and nothing will be printed. In other words, to see the debug log produced by Easy Mobile (or any other scripts), you have to create development builds (to create development builds, check the "Development Build" option in the Build Settings dialog).

Excluding debug logging in release builds is a common practice in app development, since it can have negative impact on the overall app performance.

# Advertising

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

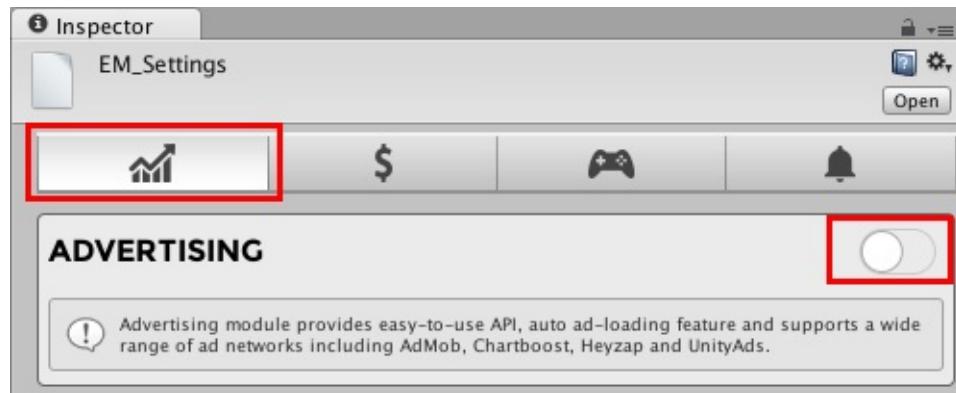
- **Supports multiple networks**
  - This module allows showing ads from most of top ad networks: AdColony, AdMob, Chartboost, Heyzap and Unity Ads
  - Even more networks can be used via Heyzap mediation
- **Using multiple networks in one game**
  - It's possible to use multiple ad networks at the same 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
AdColony		●	●
AdMob	●	●	●
Chartboost		●	●
Heyzap	●	●	●
Unity Ads		●	●

# Module Configuration

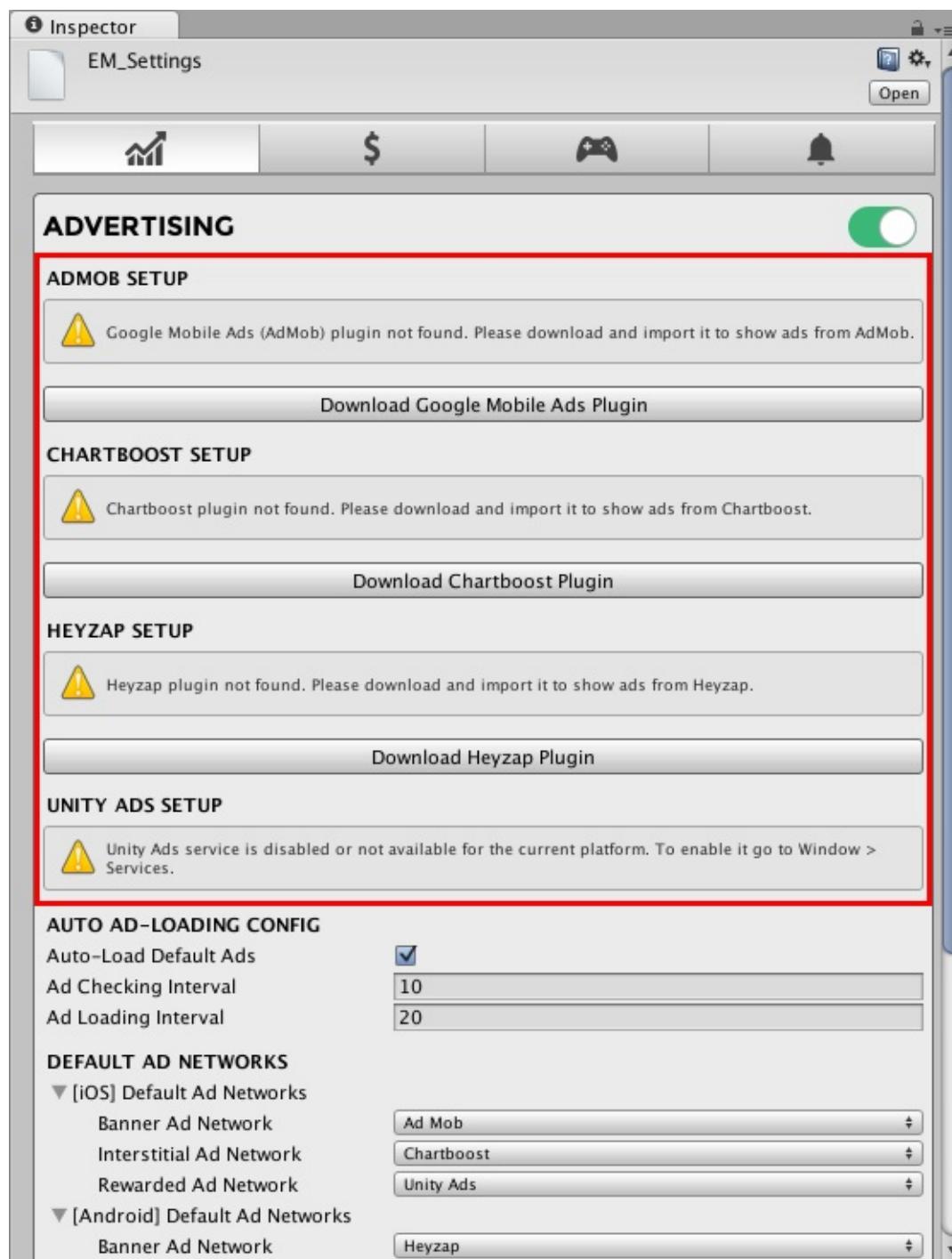
To use the Advertising module you must first enable it. Go to *Window > Easy Mobile > Settings*, select the Advertising tab, then click the right-hand side toggle to enable and start configuring the module.



# 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.



## A Note on using Heyzap

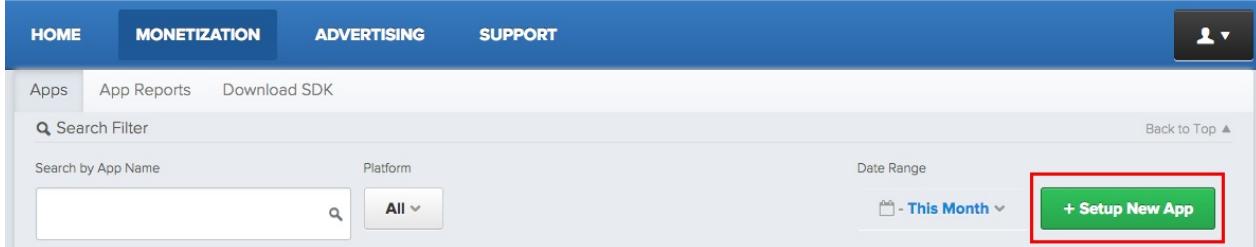
Heyzap can serve ads from multiple other networks thanks to its mediation feature. To do so it requires 3rd-party SDKs to be imported together with its own plugin. If you use Heyzap, you should not import the standalone-plugins of other networks (AdColony, AdMob, Chartboost, etc.), to avoid potential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

# AdColony

## Create AdColony Apps and Zone Ids

To show ads from AdColony you need to create apps and ad zones in its clients portal. To access the clients portal, create an account and login to [AdColony page](#).

In the clients portal, select **MONETIZATION** tab, then select the **Apps** sub-tab and click the **Setup New App** button.



In the opened page enter the required information for your new app, e.g. app name, platform and location. You can also select the ad types that you would like to allow in your app. Hit **Create** when you're done, your app will be created and you'll be redirected back to the **Apps** page. Select your newly created app to reveal its information, which looks similar to the picture below. Note the **AdColony App ID** as we will use it later.

 A screenshot of the AdColony app edit page for an app named "Easy Mobile Demo (Unreleased)". The page has a header "Basic App Information". Below it is a table with the following data:
 

App Name	Easy Mobile Demo (Unreleased)
Platform	
Status	3 Zones
Price	Free
Categories	Casual,
AdColony App ID	app770e6afa48

 The "AdColony App ID" field is highlighted with a red rectangular box.

Now your app is ready, the next step is to create ad zones for it. Click the **Setup New Ad Zone** at the bottom of the app edit page to create a new ad zone.

In the **Integration** section, give your ad zone a name, optional notes and set its as active. Note the **Zone ID** as we'll use it later.

The Zone ID will appear once you save your new ad zone.

## Integration

---

Zone is active?

Yes  No

Zone ID: **vz9494457a075**

Name your ad zone

rewarded-ad

Special notes on this zone

Dedicated zone for rewarded ads

In the **Creative Type** section, select the **Video** option.

## Creative Type

---

WARNING: Creative type for a zone cannot be change once the zone is created

Video

Display (In Testing)

In the **Zone Type** section, select **Preroll/Interstitial** if you want to use this zone for interstitial video ads. Otherwise, select **Value Exchange/V4VC** to use it for rewarded ads.

## Zone Type

---

Preroll/Interstitial ?

Value Exchange/V4VC ?

V4VC Secret Key: **v4vc725c8f3386**

Client Side Only?

Yes  No

Virtual Currency Name

Credits

Daily Max Videos per User

20

Must be greater than 0

Reward Amount

1

Must be greater than 0

In the **Options** section, you can set a daily cap or a session cap to limit the number of ads served to a user per day or per session, respectively. In the **Development** section, you can choose to show test ads only (for debug purpose), don't forget to disable this option when your app is released.

### Options

Daily play cap	Session play cap
<input type="text" value="0"/> (0 for no limit)	<input type="text" value="0"/> (0 for no limit)
<input type="checkbox"/> Enable Ad Skipping After	<input type="text" value="0"/> Seconds (This setting allows users to skip ads after a delay.) <a href="#">?</a>
<input type="checkbox"/> Override App-Level Settings	

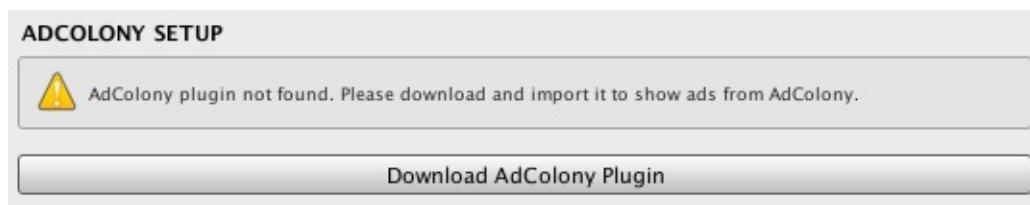
### Development

Show test ads only (for dev or debug)?  Yes  No

Now your new ad zone is fully configured, click the **Save** button to save it. Repeat the process to create other ad zones to suit your needs. Typically, you'd want to have 2 ad zones, one for interstitial ads and one for rewarded ads. If you're targeting multiple platforms, create a new app for each platform, and for each app create the necessary ad zones.

## Import AdColony Plugin

To have your Unity app work with AdColony you need to import the [AdColony plugin for Unity](#). In the **ADCOLONY SETUP** section of the Advertising module, click the *Download AdColony Plugin* button to open the download page. Download the plugin and import it to your project.



## Configure AdColony

After importing the AdColony plugin, the **ADCOLONY SETUP** section will be updated as below.

**ADCOLONY SETUP**

(!) AdColony plugin was imported.

[Download AdColony Plugin](#)

▼ [iOS] AdColony Ids

App Id	
Interstitial Ad Id	
Rewarded Ad Id	

▼ [Android] AdColony Ids

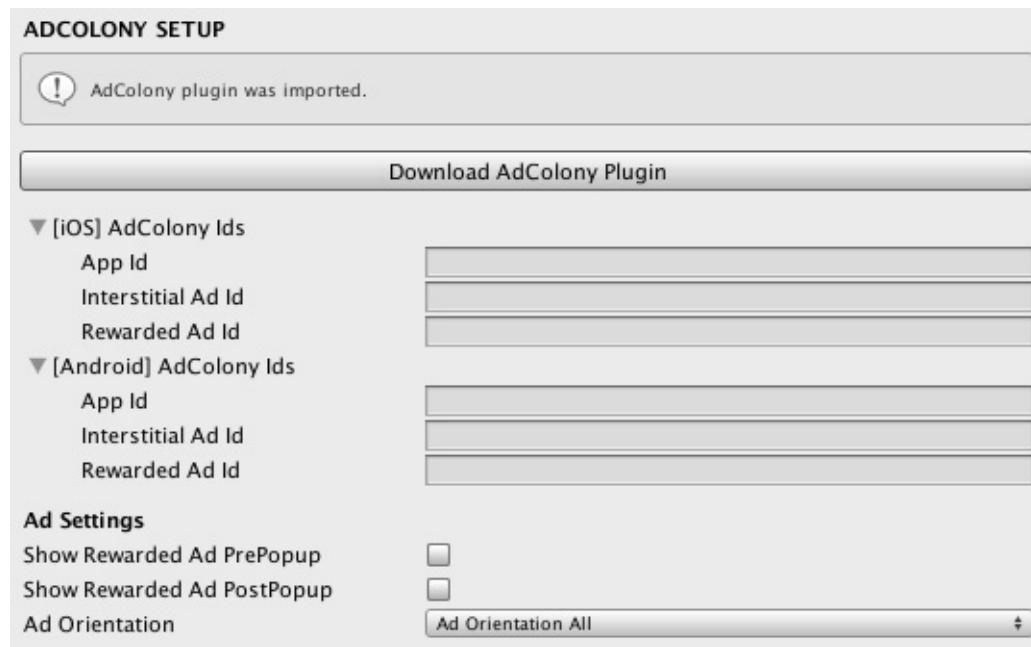
App Id	
Interstitial Ad Id	
Rewarded Ad Id	

**Ad Settings**

Show Rewarded Ad PrePopup

Show Rewarded Ad PostPopup

Ad Orientation

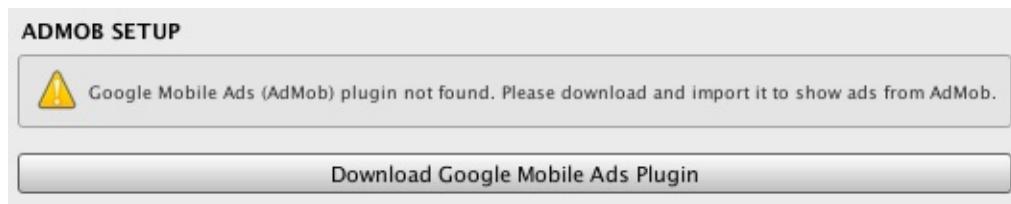


- *[iOS] AdColony Ids*: enter the app ID and zone IDs created in the AdColony clients portal for the iOS app
- *[Android] AdColony Ids*: enter the app ID and zone IDs for the Android app
- *Show Rewarded Ad PrePopup*: show the AdColony's default popup before a rewarded video starts
- *Show Rewarded Ad PostPopup*: show the AdColony's default popup after a rewarded video has finished
- *Ad Orientation*: select the orientation for the ads to match your app settings

# AdMob

## Import AdMob Plugin

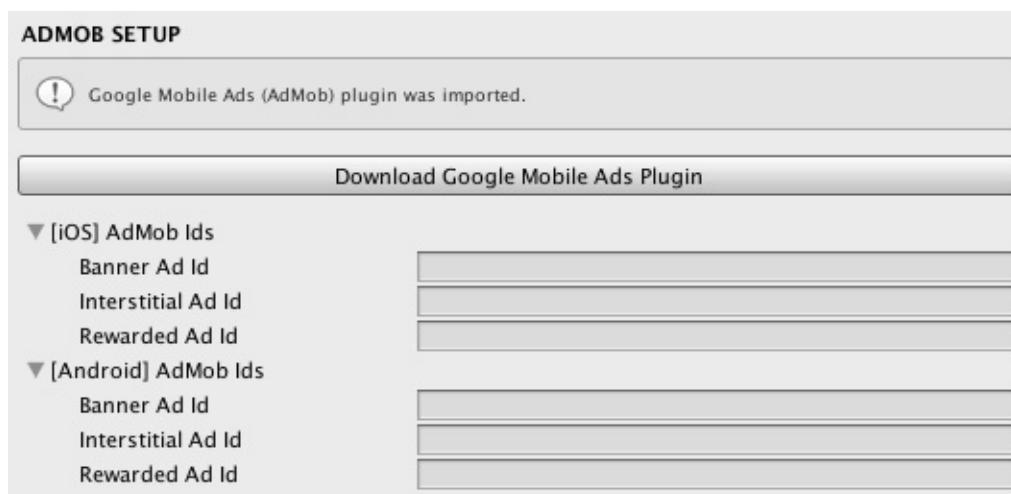
To show ads from AdMob you need to import the [Google Mobile Ads plugin](#). In the **ADMOB SETUP** section, click the *Download Google Mobile Ads Plugin* button to open the download page. Download the plugin and import it to your project.



## Configure AdMob

### Enter Ad IDs

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

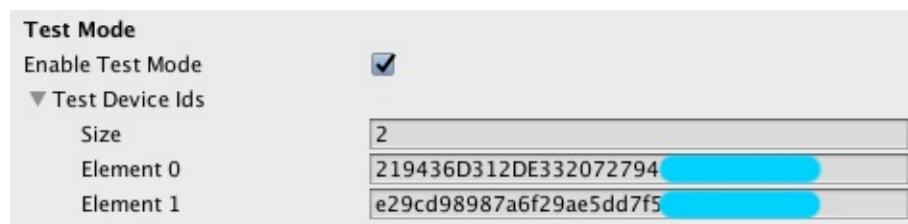


Note that you only need to provide the IDs of the ad units you want to use, e.g. if you only use AdMob banner ad you can leave the interstitial ad and rewarded ad IDs empty.

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-0664570763252260xxxxxxxxxxxx.

## Using AdMob Test Mode

To enable AdMob's test mode, simply check the *Enable Test Mode* option and enter the IDs of your testing devices into the *Test Device Ids* array.



You can find the ID of your test device by building and running the Easy Mobile demo app on that device. Remember to add the EasyMobile prefab to the DemoHome scene before starting the build.

### Android device ID

- In Unity, build the Easy Mobile demo app for Android platform
- Install and run the demo app on your testing device
- Open Terminal (Mac) or Cmd (Windows) and type in

```
adb logcat -s Ads
```

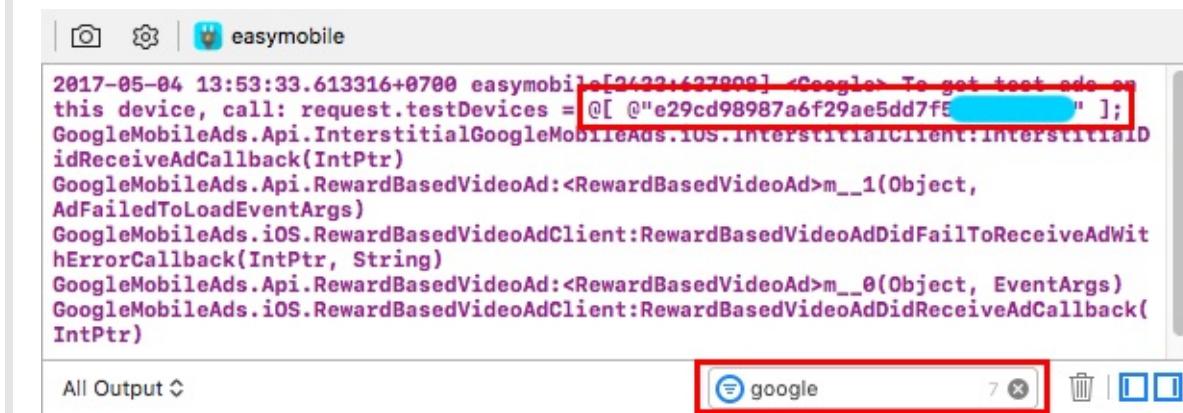
(if you're on Windows, you may need to [add the Android SDK path to the Windows System PATH](#))

- In the demo app, select ADVERTISING and then click the SHOW BANNER AD button
- Observe the output logcat in the Terminal/Cmd and locate a line similar to the one in the following image, the value between the double quotes is your device ID

```
I Ads      : Starting ad request.  
I Ads      : Use AdRequest.Builder.addTestDevice("219436D312DE332072794[REDACTED]") to get test ads on this device.
```

### iOS device ID

- In Unity, build the Easy Mobile demo app for iOS platform
- Open the generated project in Xcode and run it on your testing device
- Type 'google' into the filter box of the Xcode Console, and find your device ID between the double quotes as highlighted in the following image



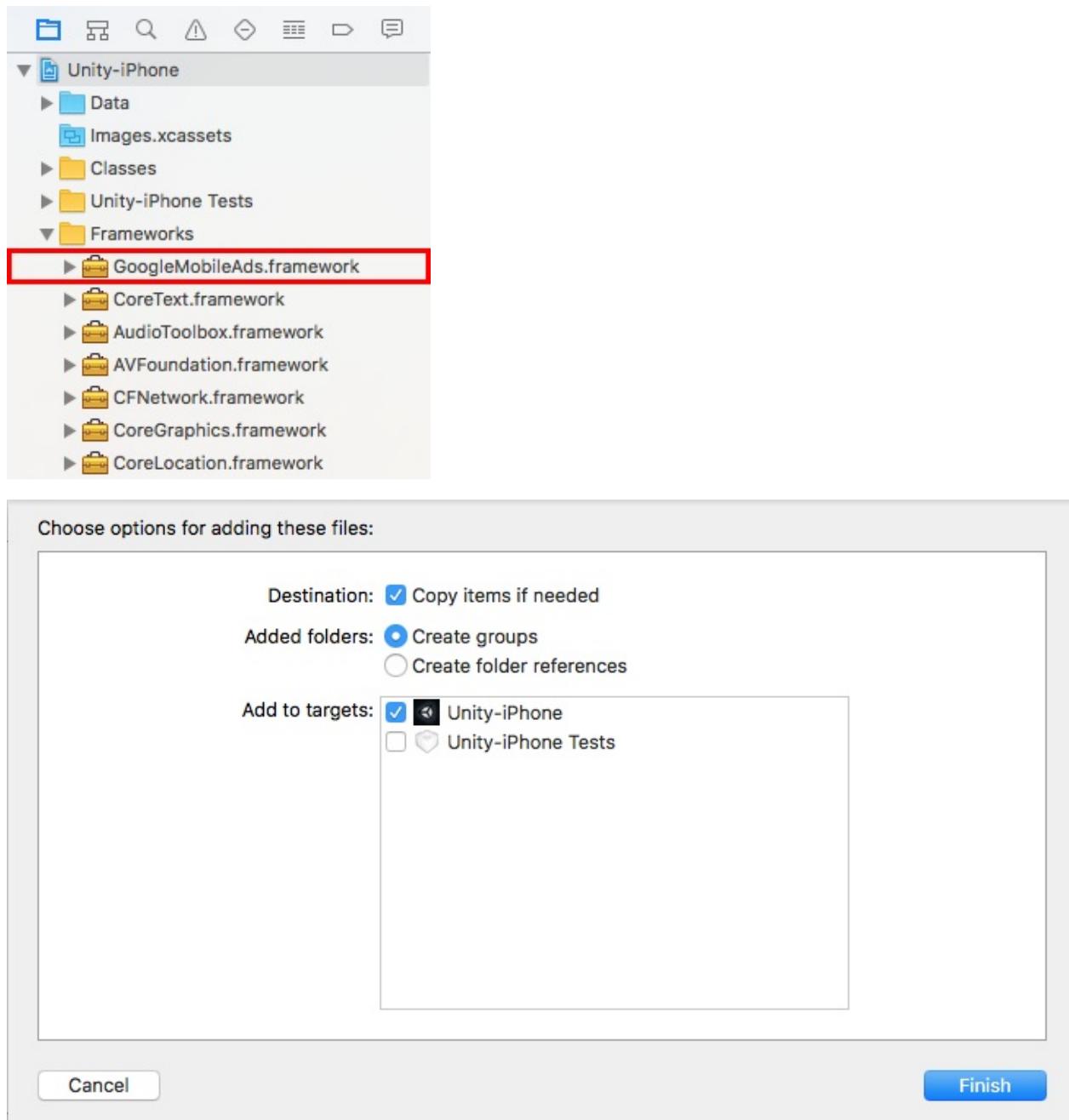
## [iOS] Import GoogleMobileAds framework in Xcode

**For Google Mobile Ads plugin version 3.1.0 or newer**

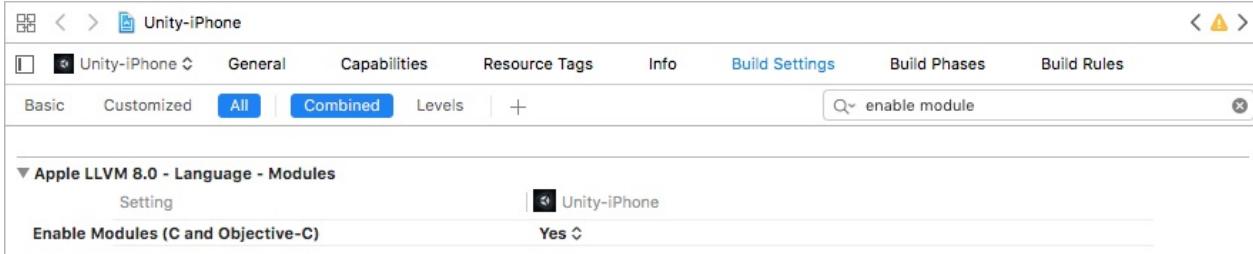
Since version 3.1.0, the Google Mobile Ads plugin for Unity has removed the CocoaPods integration, therefore after creating an iOS build in Unity, you need to manually import the GoogleMobileAds framework for iOS to the generated Xcode project, or it will fail to build due to file missing. Please follow the following steps to import this required framework to your Xcode project.

First you need to download the GoogleMobileAds framework from  
<https://firebase.google.com/docs/admob/ios/download>

Next, unzip the downloaded file, and drag the **GoogleMobileAds.framework** to the *Frameworks* folder in your Xcode project. Also check the "Copy items if needed" option in the import dialog.

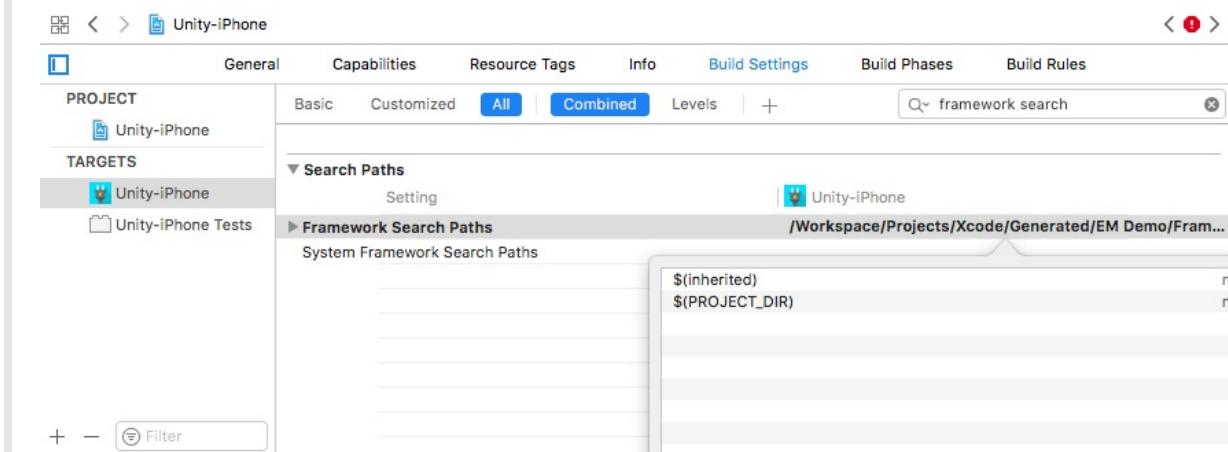


Lastly, select the Unity-iPhone project in the project navigator, then select the Unity-iPhone target in the **TARGETS** list, open *Build Settings* tab and set the **Enable Modules (C and Objective-C)** option to **Yes**.



Note that if you select **Replace** in a subsequent iOS build in Unity, you'll need to repeat the above process to import the GoogleMobileAds framework again.

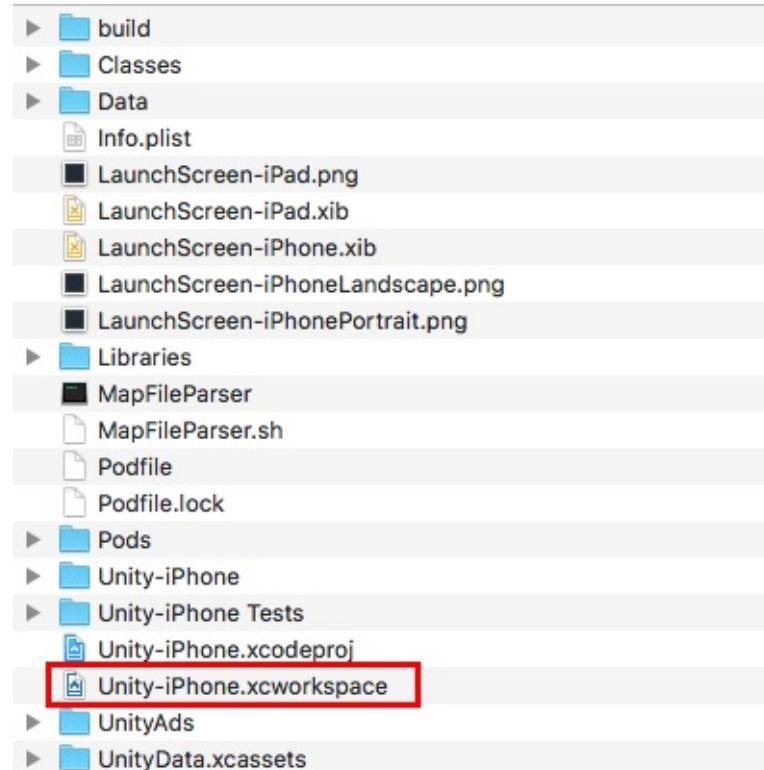
If you select **Append**, you don't need to reimport the framework. However, you may get the error "**Module 'GoogleMobileAds' not found**" when building in Xcode, which you can fix by adding **\$(PROJECT\_DIR)** to the **Framework Search Paths** key in *Build Settings*.



## For Google Mobile Ads plugin versions older than 3.1.0

Before version 3.1.0, the Google Mobile Ads plugin for Unity employs CocoaPods to automatically import the GoogleMobileAds framework to the generated Xcode project when an iOS build is performed in Unity. Therefore you need to install CocoaPods to your Mac if you intend to use a pre-3.1.0 version of the Google Mobile Ads plugin. Please go to <https://cocoapods.org/> for install instructions, as well as for more information about CocoaPods. Note that you only need to install CocoaPods, everything else will be done automatically by the Google Mobile Ads plugin.

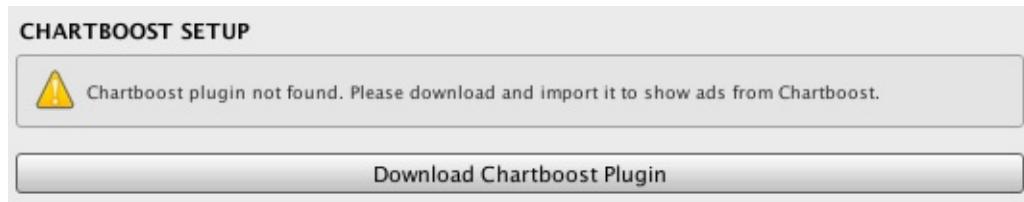
When building for iOS in Unity, CocoaPods will automatically create an Xcode workspace (with .xcworkspace extension) in the generated Xcode project. You should always open this workspace instead of the normal project file (with .xcodeproj extension).



# Chartboost

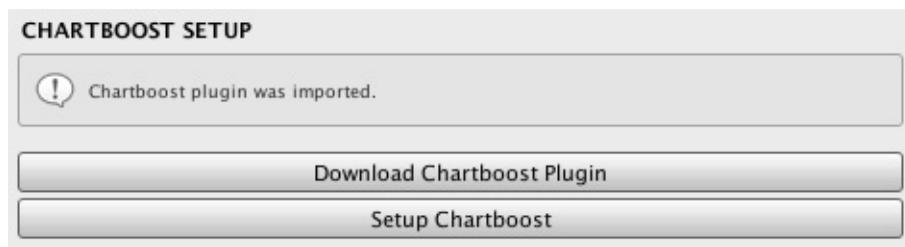
## Import Chartboost Plugin

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.

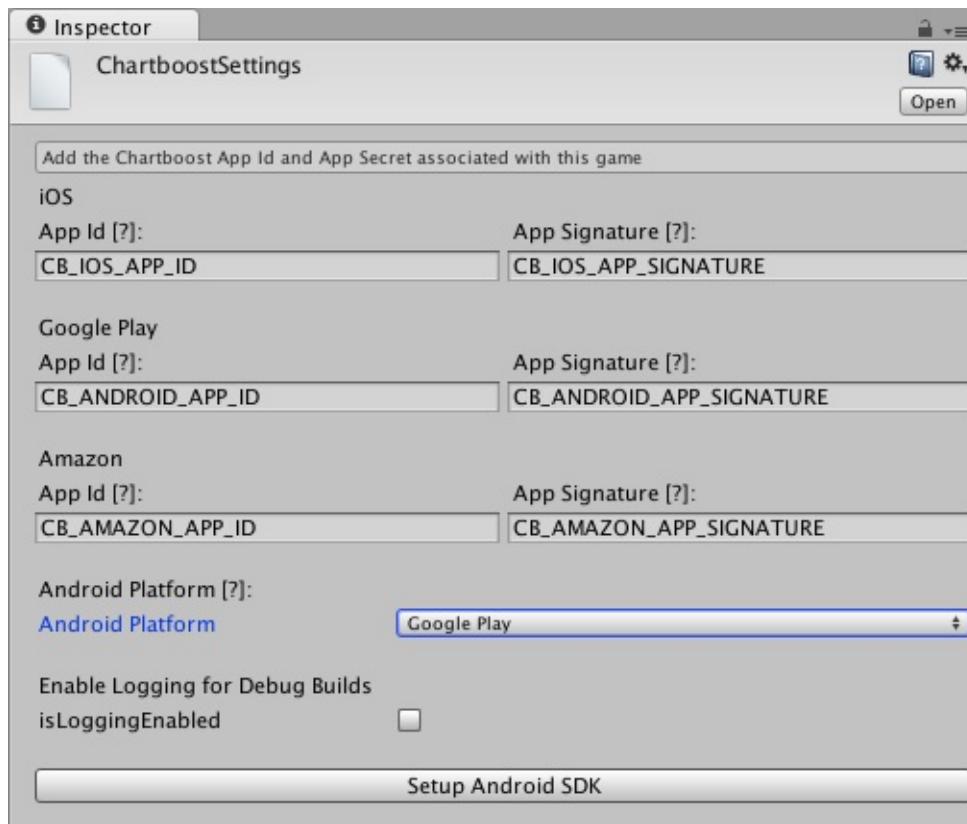


## Configure Chartboost

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



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



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 screenshot shows the 'App Settings' page in the Chartboost dashboard. On the left, there's a sidebar with options like Overview, Analytics, Campaigns, App Settings (selected), and Tools. The main area has fields for Platform (set to iOS), App Nickname (Buster's Boost), App Bundle ID (com.chartboost.apollo), and App Orientation (Portrait and Landscape selected). Below these, there's a note about selecting a default orientation. To the right, there's a summary box containing the App ID (5689b44f8838092b63cf94f0) and App Signature (edabc1ce4ad0318edd64a4584e59b534c985373c). A large green arrow points from the 'Import App' button towards this summary box.

## **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.

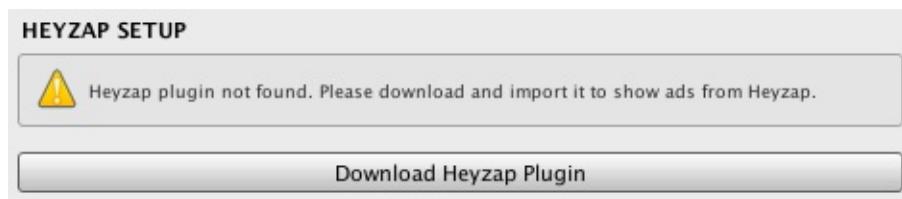
- To create a publishing campaign follow the instructions [here](#)
- To enable Test Mode follow the instruction [here](#)

# Heyzap

As mentioned earlier, if you use Heyzap's mediation with other networks (AdColony, AdMob, Chartboost, etc.), you should not import the standalone-plugins of those networks, to avoid potential conflicts. Instead, import their corresponding packages provided at the Heyzap download page.

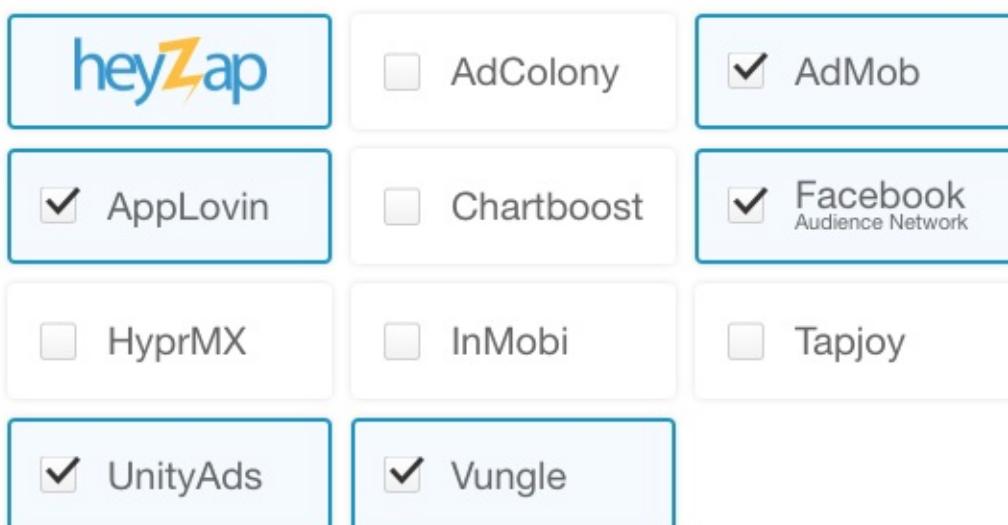
## Import Heyzap Plugin

To show ads from Heyzap you need to import the [Heyzap plugin for Unity](#). In the **HEYZAP 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:

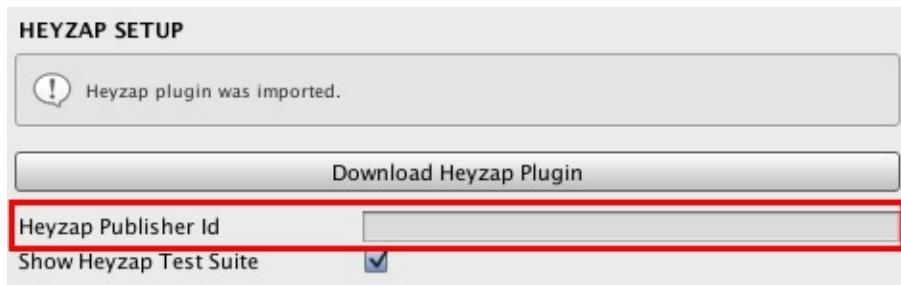


Follow the instructions provided by Heyzap to download and import its plugin as well as other required 3rd-party plugins. Also go through the **Integration Notes** section below to avoid problems that may occur during the integration of 3rd-party networks.

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

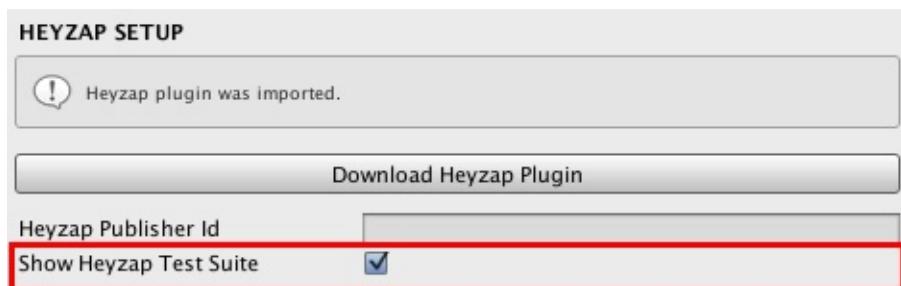
## Configure Heyzap

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.

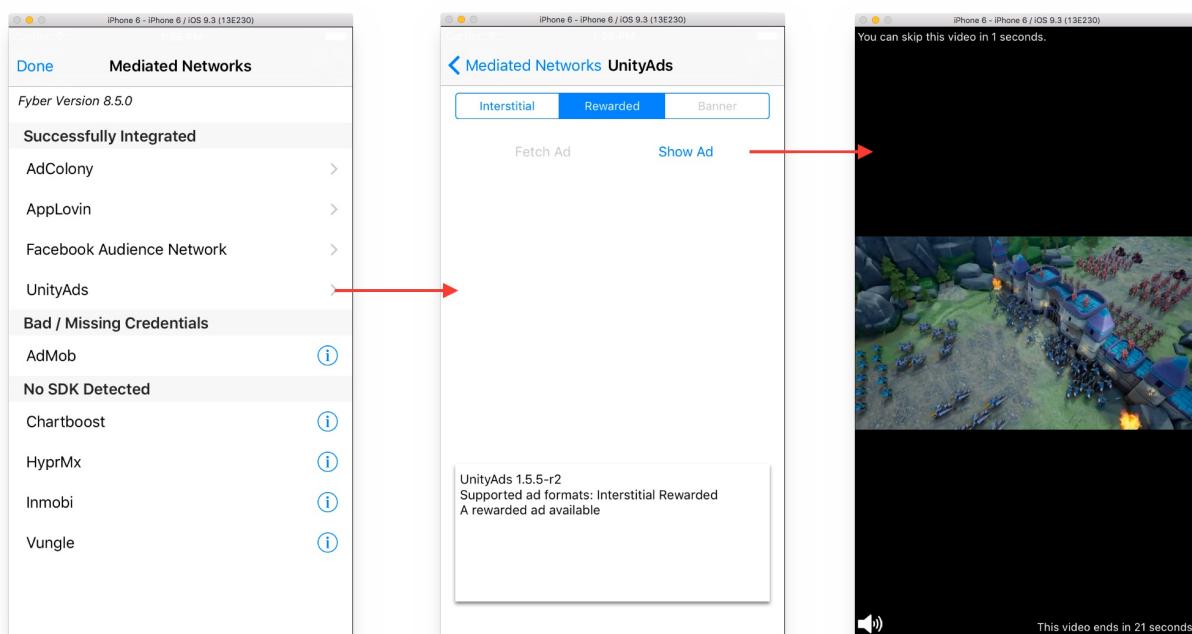


## Heyzap Mediation Test Suite

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.



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.

### 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)

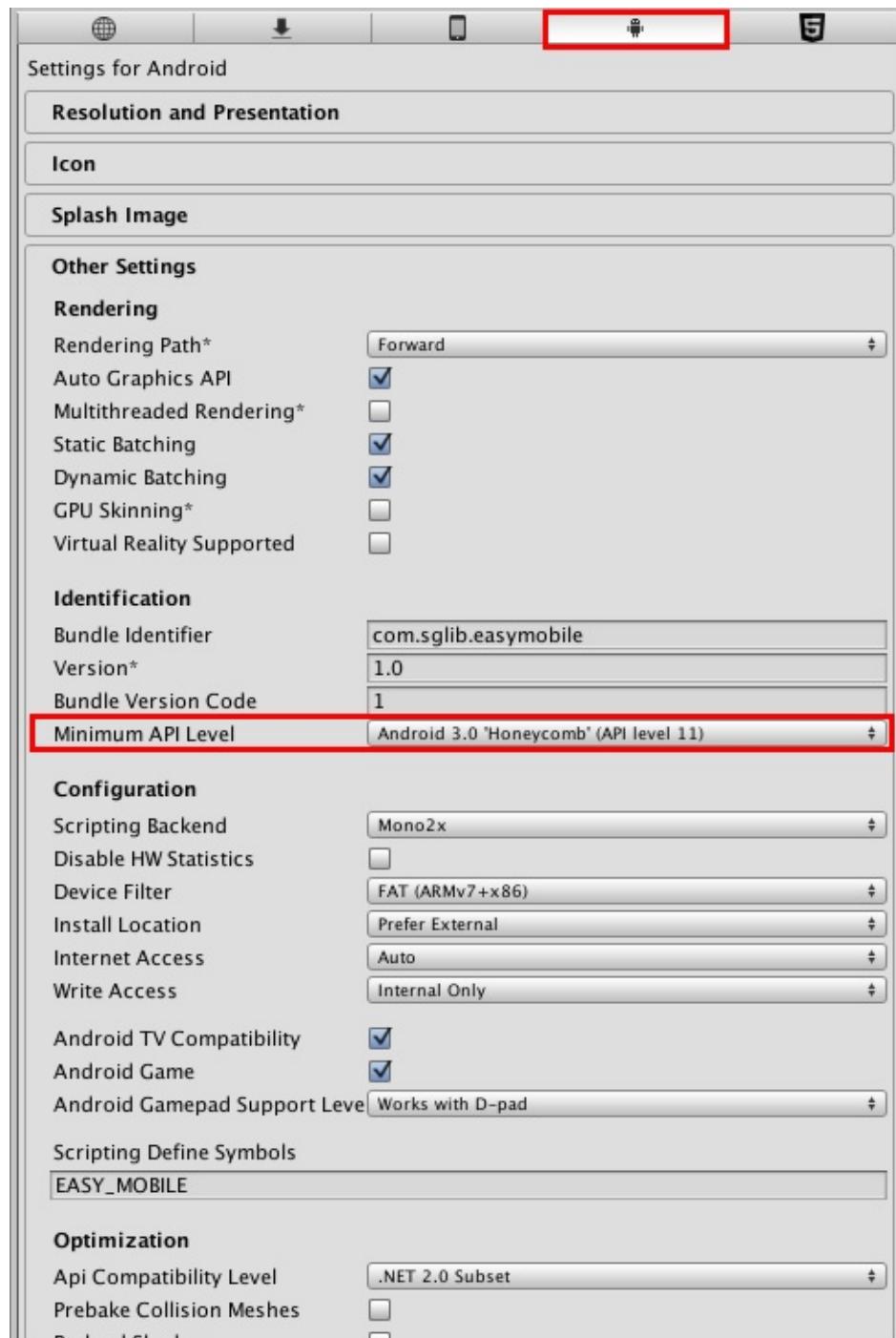
The current Heyzap SDK requires a *minSdkVersion* of 10, while some other 3rd-party plugins may require a version of 11 or above. If you get a build error including this line

```
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

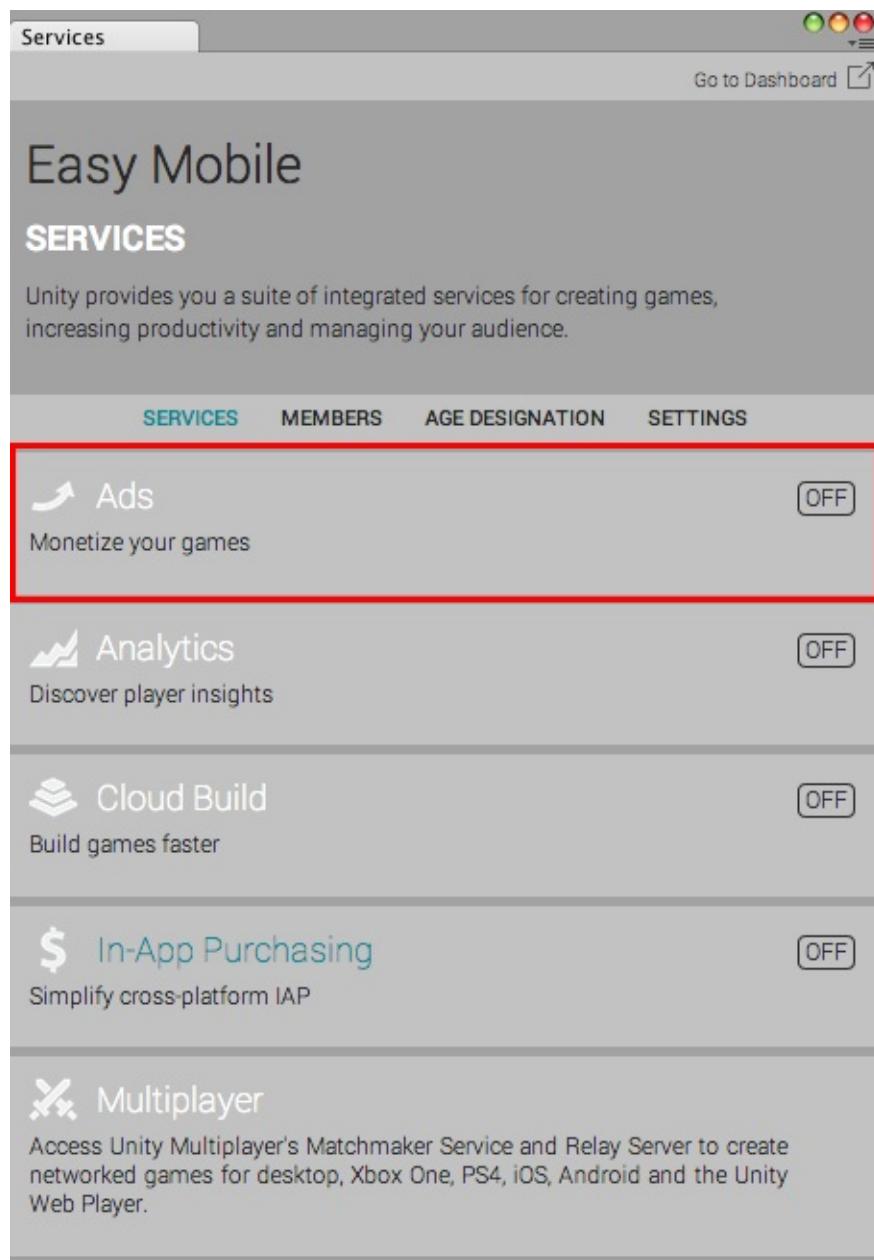
## Enable Unity Ads Service

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

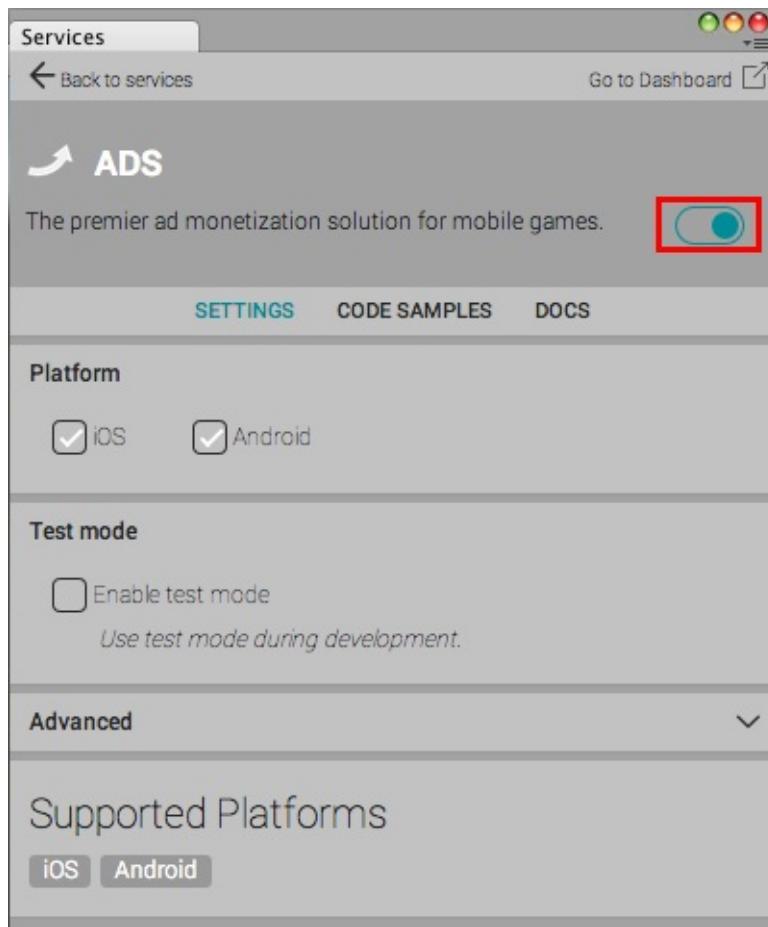
To show ads from Unity Ads you need to enable the corresponding service. Easy Mobile will automatically check for the service's availability and warn you to enable it if needed. Below is the **UNITY ADS SETUP** section when Unity Ads is not enabled.



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

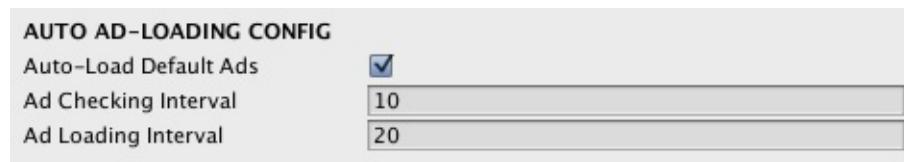
It is advisable to enable the test mode of Unity Ads during development. This will ensure there's always an ad returned whenever requested. To enable test mode simple check the *Enable test mode* option within the Ads configuration window.

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-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

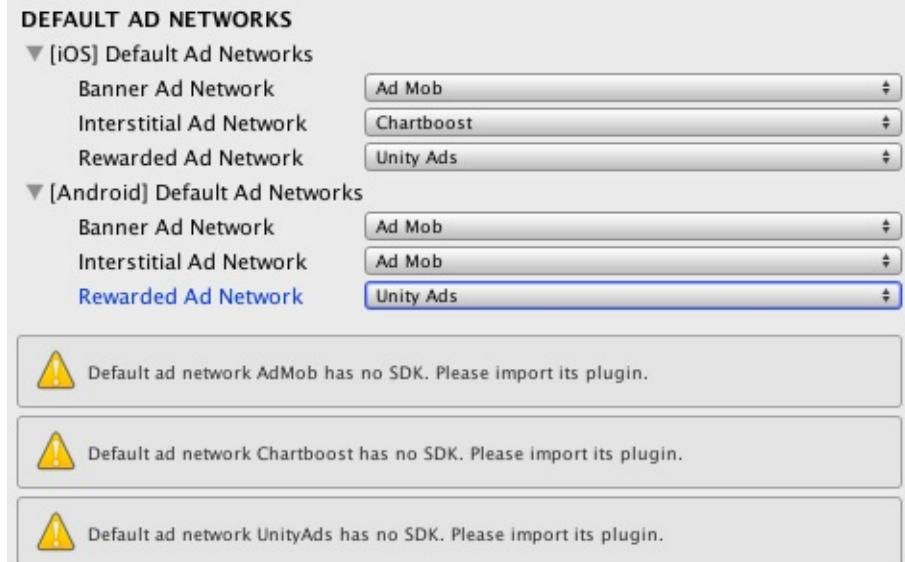
▼ [Android] Default Ad Networks

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();
```



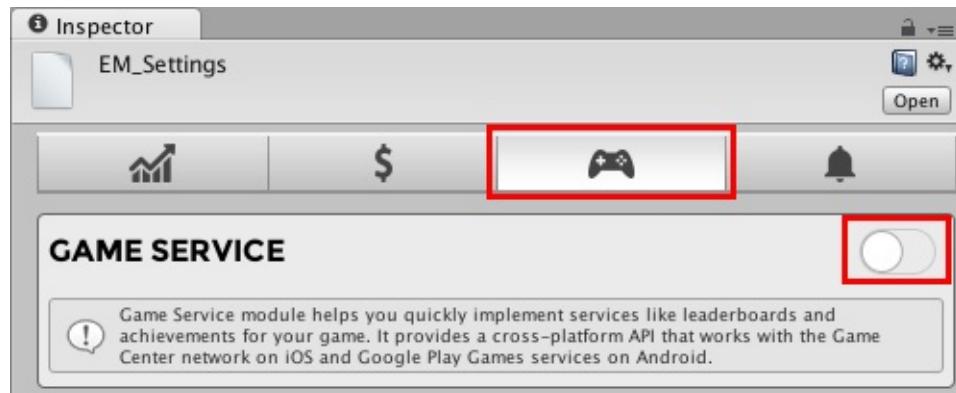
# Game Service

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.

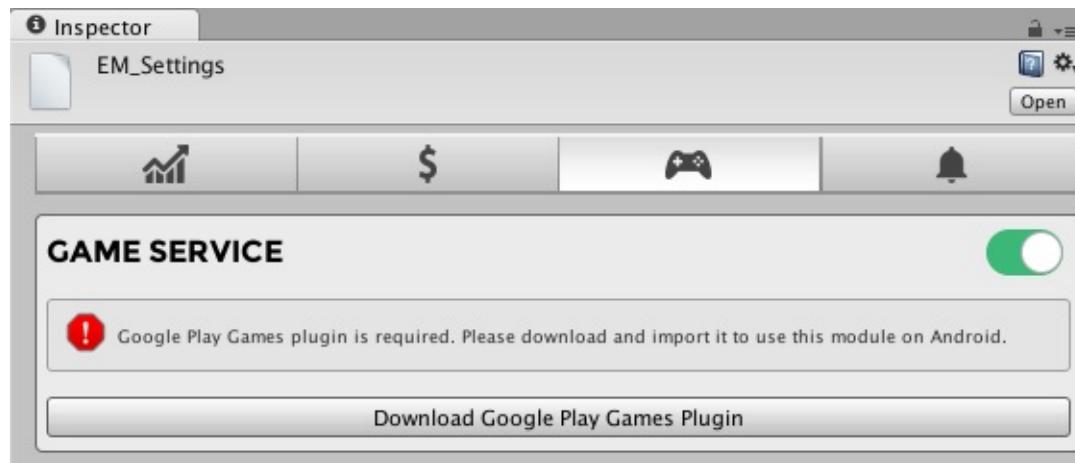


# Android-Specific Setup

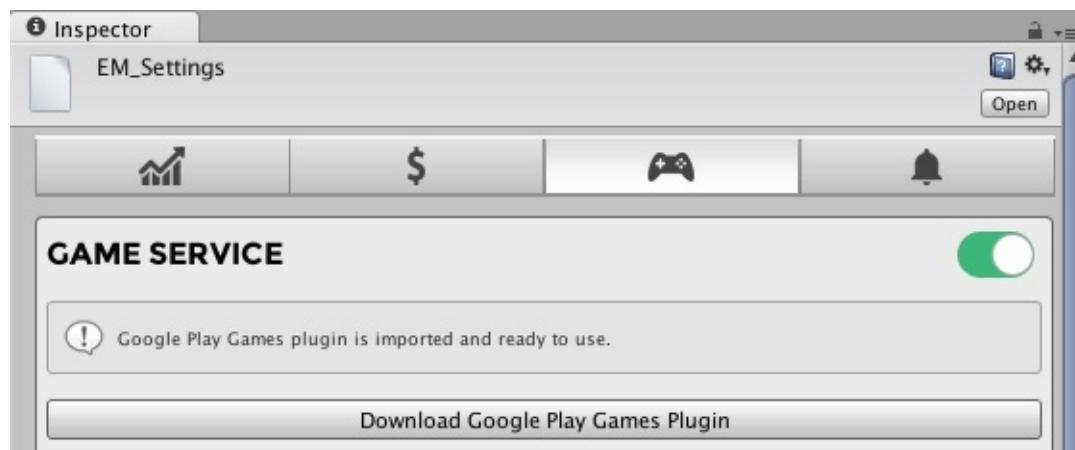
## Import Google Play Games plugin for Unity

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.



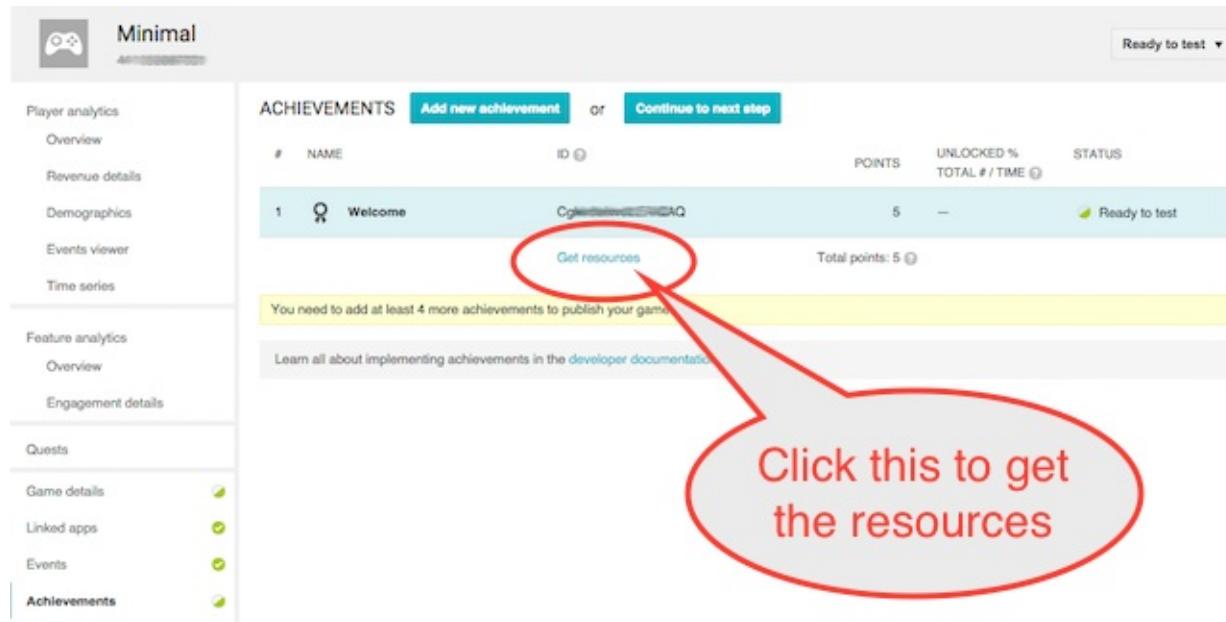
Since we're not using Google Play Games plugin on iOS, the NO\_GPGS symbol will be defined for iOS platform automatically after the plugin is imported in order to disable it.

## Setup Google Play Games

To setup Google Play Games plugin, you need to obtain the game resources from the Google Play Developer Console.

The game resources are available after you configured your game on the Google Play Developer Console. If you're not familiar with the process, please follow the instructions on creating a [client ID](#), as well as [leaderboards](#) and [achievements](#).

To get the game resources, login to your Google Play Developer Console, select *Game services* tabs then select your game. Next go to the *Achievements* tab and click on the *Get Resources* label at the bottom of the list.



Copy all the xml content from the *Android* tab.

## EXPORT RESOURCES

**Android**   **Objective-C**   **Javascript**   **Text**

```
<?xml version="1.0" encoding="utf-8"?>
<!--
Google Play game services IDs.
Save this file as res/values/games-ids.xml in your project.
-->
<resources>
<string name="app_id">41[REDACTED]31</string>
<string name="package_name">com.[REDACTED].minimal</string>
<string name="achievement_welcome">CgkIgrCK24cTEAIQAw</string>
</resources>
```

**Done**

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*.

**[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**

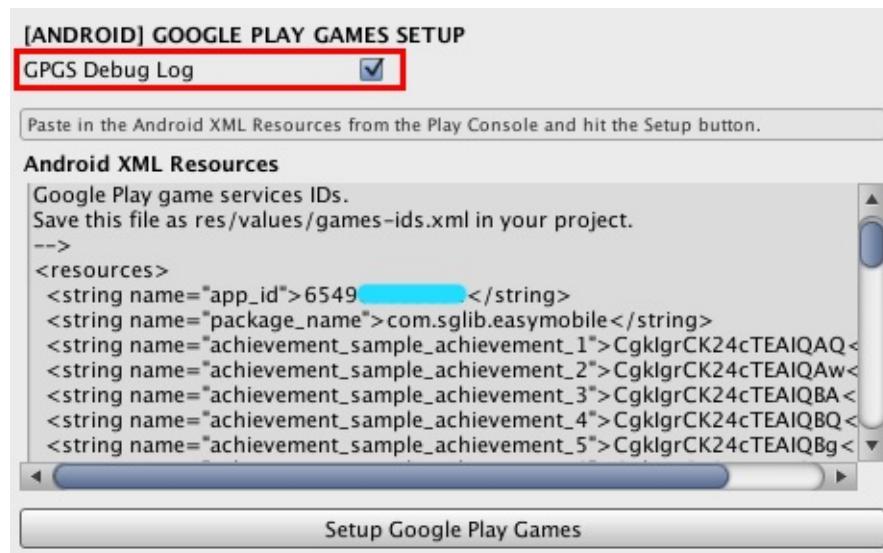
```
<?xml version="1.0" encoding="utf-8"?>
<!--
Google Play game services IDs.
Save this file as res/values/games-ids.xml in your project.
-->
<resources>
<string name="app_id">6549[REDACTED]</string>
<string name="package_name">com.sgilib.easymobile</string>
<string name="achievement_sample_achievement_1">CgkIgrCK24cTEAIQAw</string>
<string name="achievement_sample_achievement_2">CgkIgrCK24cTEAIQAw</string>
<string name="achievement_sample_achievement_3">CqklqrCK24cTEAIQBA</string>
```

**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.



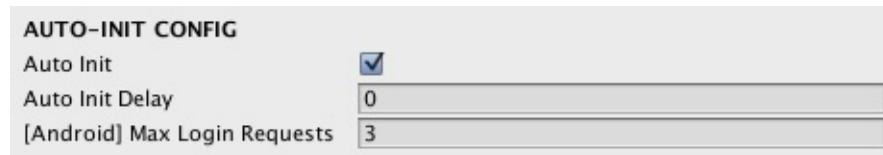
# Auto Initialization

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*: uncheck this option to disable the auto initialization feature, you can start the initialization manually from script (see the **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 MonoBehavior (attached to the EasyMobile prefab) runs.

# Leaderboards & Achievements

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

## Before You Begin

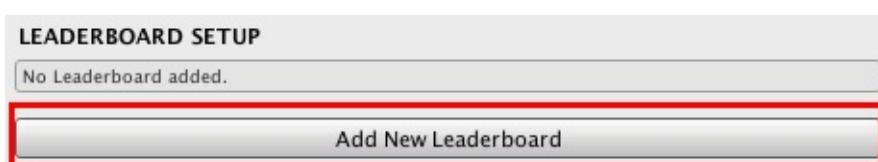
It is assumed that you already configured your game for the targeted gaming networks, i.e. Game Center and Google Play Games. If you're not familiar with the process, here're some useful links:

- 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](#)

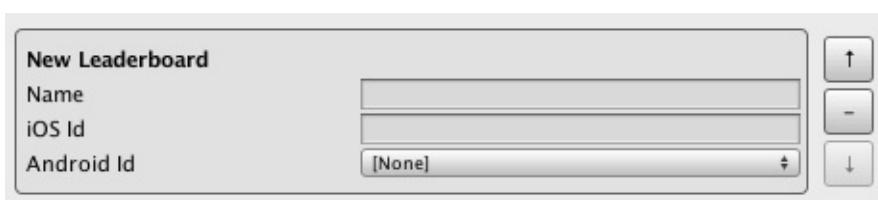
In the **LEADERBOARD SETUP** and **ACHIEVEMENT SETUP** you can add, edit or remove leaderboards and achievements.

## 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.

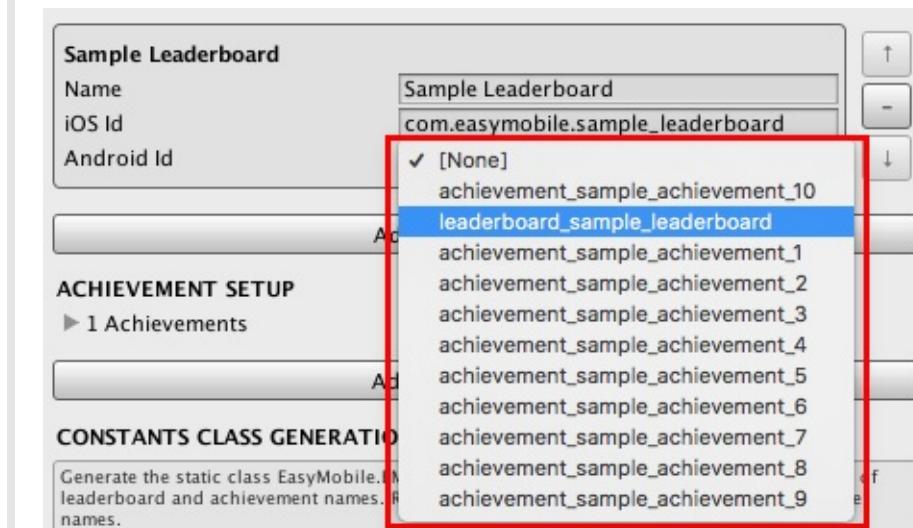


Fill in the required information of the leaderboard (or achievement):

- *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

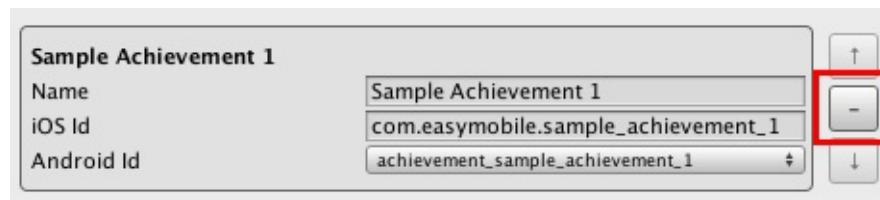
- *Android Id*: the ID of this leaderboard (or achievement) as declared in Google Play Developer Console

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\_GPGSIds file?), allowing Easy Mobile to show a nice dropdown of all defined leaderboard and achievement IDs for you to choose from.



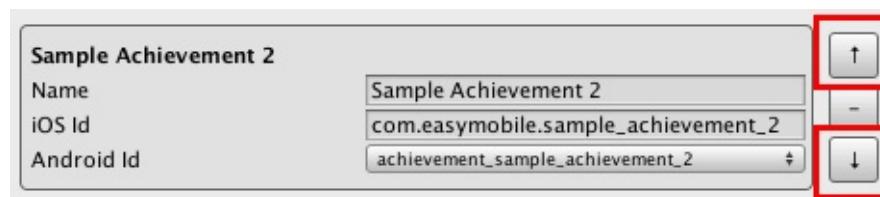
## Remove a Leaderboard or Achievement

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.

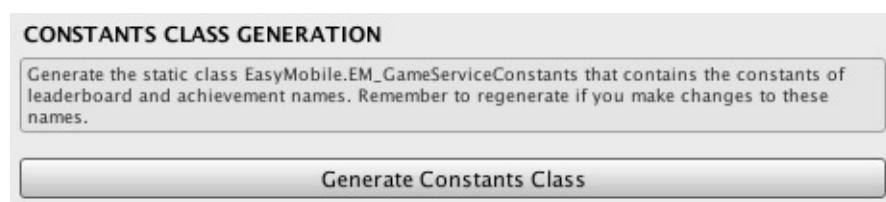




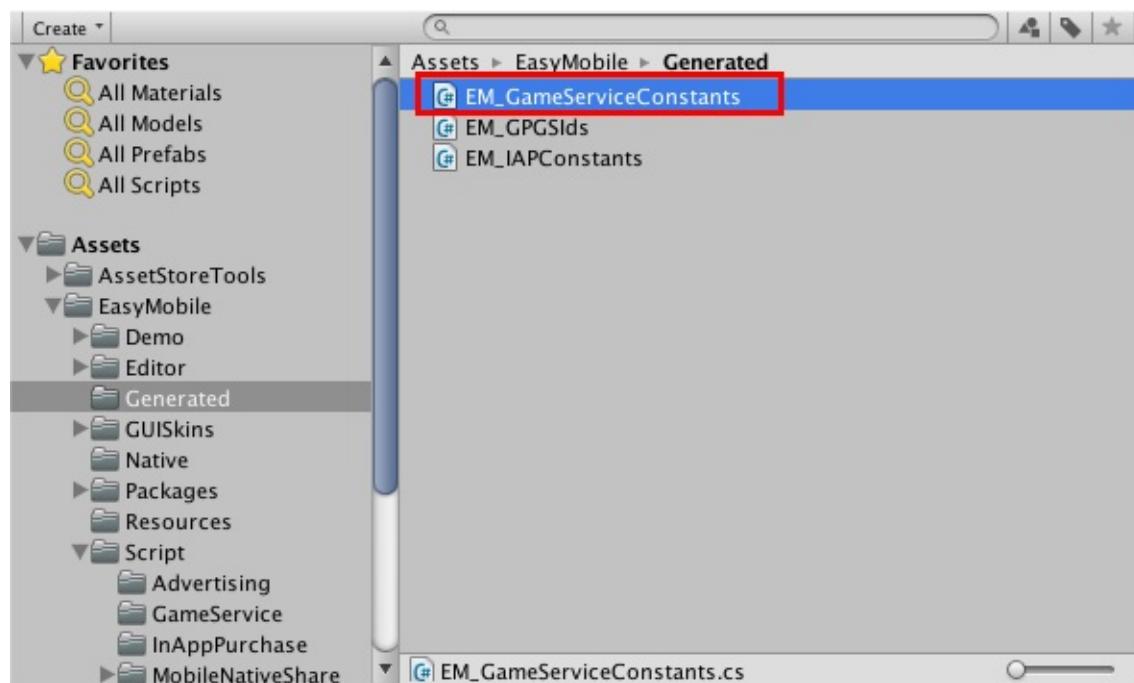
# Game Service Constants Generation

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 other action, e.g. reporting scores, can be done. 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...
}
```

A *UserLoginSucceeded* event will be fired when the initialization completes and the user logsins successfully. Otherwise, a *UserLoginFailed* event will be fired instead. You can optionally subscribe to these events and take appropriate actions depended on the user login status.

```
// Subscribe to events in the OnEnable method of a MonoBehavior script
void OnEnable()
{
    GameServiceManager.UserLoginSucceeded += OnUserLoginSucceeded;
    GameServiceManager.UserLoginFailed += OnUserLoginFailed;
}

// Unsubscribe
void OnDisable()
{
    GameServiceManager.UserLoginSucceeded -= OnUserLoginSucceeded;
    GameServiceManager.UserLoginFailed -= OnUserLoginFailed;
}

// Event handlers
void OnUserLoginSucceeded()
{
    Debug.Log("User logged in successfully.");
}

void OnUserLoginFailed()
{
    Debug.Log("User login failed.");
}
```

You can also check if the module has been initialized at any point using the *IsInitialized* method.

```
// 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 scope
// 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 rogress of 50% for an incremental achievement
// EM_GameServiceConstants.Sample_Incremental_Achievement is the generated name consta
nt
// of an incremental achievement named "Sample Incremental Achievement"
GameServiceManager.UnlockAchievement(EM_GameServiceConstants.Sample_Incremental_Achiev
ement, 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 sett
ings
// 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.");
    }
}
```

## Sign Out

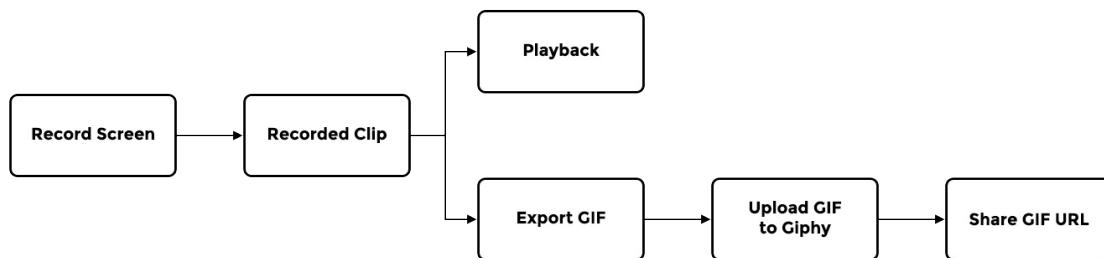
To sign the user out, simply call the *SignOut* method. Note that this method is only effective on Android.

```
// Sign the user out on Android
GameServiceManager.SignOut();
```



# GIF

The GIF module provides you convenient tools to record screen activities into a short clip, play the recorded clip and export it into a GIF image. You can then upload the GIF file to hosting sites like [Giphy](#) and finally share its URL to social networks. In short, this module helps you easily add the GIF sharing feature to your game, which allows the user to share animated GIF images of the gameplay, instead of still screenshots, to social networks including Facebook and Twitter. The following picture illustrates a typical workflow of such feature.



Here're some highlights of this module:

- **High performance, mobile-friendly GIF generator**
  - Low overhead screen/camera recorder
  - GIF generation is done in native code (iOS and Android) on a separate thread to allow fast exporting while minimizing impact to the main thread. Export callbacks are still called from main thread though, so you can safely access Unity API in the callback handlers
- **Flexible, fully controllable process**
  - You have full control on the sizes, length, frame rate, loop mode and quality of the exported GIF
  - You can also set the priority of the exporting thread to best suit your needs
- **High quality GIF**
  - Exported GIF employs GIF89a format and uses 256-color local palettes (one palette per frame)
  - Frame image data is [LWZ compressed](#)
- **Works in Unity editor**
  - GIF exporting also works in the editor, mostly for testing purpose. On mobiles, the

exporting is done in native code, while in editor it is done in managed code using an adapted version of the Moments plugin (see the **Acknowledgement** section)

- **Easy GIF sharing**

- This module also provides Giphy API for uploading GIF images to Giphy, so that they can be shared and played on major social networks including Facebook and Twitter using the Giphy hosted URLs

## Acknowledgement

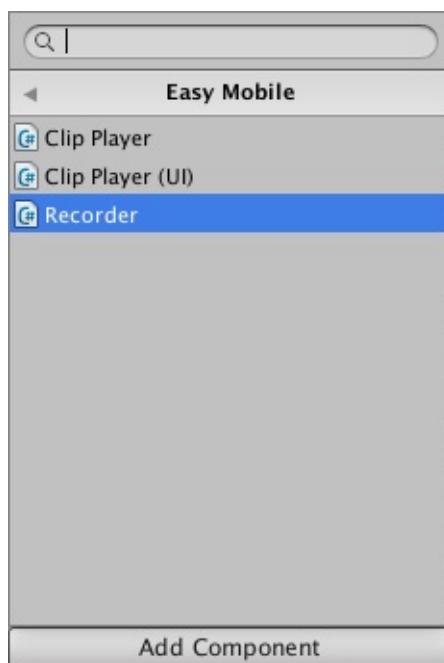
The recorder used in this module is adapted from the recorder of the [Moments plugin](#) by Chman (Thomas Hourdel). Also, in Unity editor, GIF generation is done using an adapted version of this plugin.

# Setup

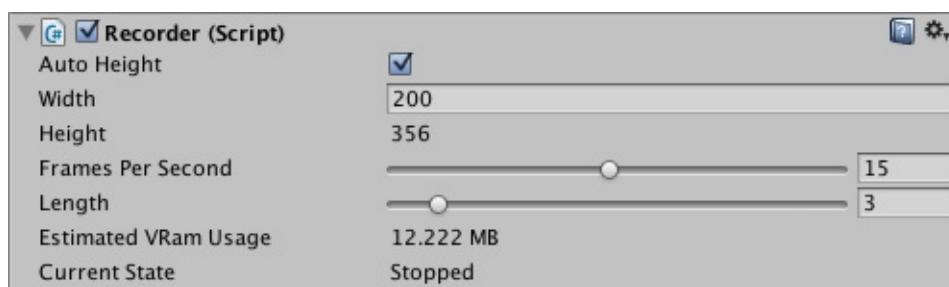
This section explains the various components, objects and concepts involved in clip recording, clip playing and GIF exporting. It also provides a guide on creating and configuring relevant objects and components.

## The Recorder component

The Recorder component records the content rendered by a camera and returns the recorded clip. To start recording, simply add a Recorder component to the camera that renders the content you're interested in recording (normally this will be the Main Camera). To add the component to a camera, select that camera in the Hierarchy, then click *Add Component > Easy Mobile > Recorder*.



Once the Recorder component is added to the camera, you can start configuring it in the inspector to determine how the recorded clip (and as a result, the exported GIF) will be like.



- **Auto Height:** whether the clip height should be computed automatically from the

specified width and the camera's aspect ratio, which is useful to make sure the exported GIF has a correct aspect ratio

- *Width*: the width of the recorded clip in pixels
- *Height*: the height of the recorded clip in pixels
- *Frames Per Second*: the frame rate of the clip
- *Length*: the clip length in seconds; the recorder automatically discards old content to preserve this length, e.g. if you set this value to 3 seconds, only last 3 seconds of the recording will be stored in the resulted clip, the rest will be discarded
- *Estimated VRam Usage*: the estimated memory used for recording, calculated based on the above settings
- *Current State*: the current status of the recorder, which is either *Stopped* or *Recording*

Now that the recorder is configured, you can start and stop its recording activity from script (see the **Scripting** section). Once the recording is stopped, the recorded clip will be returned for playback of GIF exporting.

### Recording the UI

To record the UI (Canvas content), you need to set the Canvas Render Mode to World Space or Screen Space - Camera, and set the Render Camera to the one containing the Recorder component in the latter case.

### Recording multiple composited cameras

If your scene contains multiple cameras being composited (using Camera.depth and Clear flags), you can add the Recorder component to the top-most camera, so it captures whatever content being composited and shown by that camera.

## The AnimatedClip class

Recorded clips are represented by the *AnimatedClip* class, which has following properties:

- *Width*: the width of the clip in pixels
- *Height*: the height of the clip in pixels
- *Frame Per Second*: the frame rate of the clip
- *Length*: the length of the clip in seconds
- *Frames*: an array of frames, each frame is a *Render Texture* object

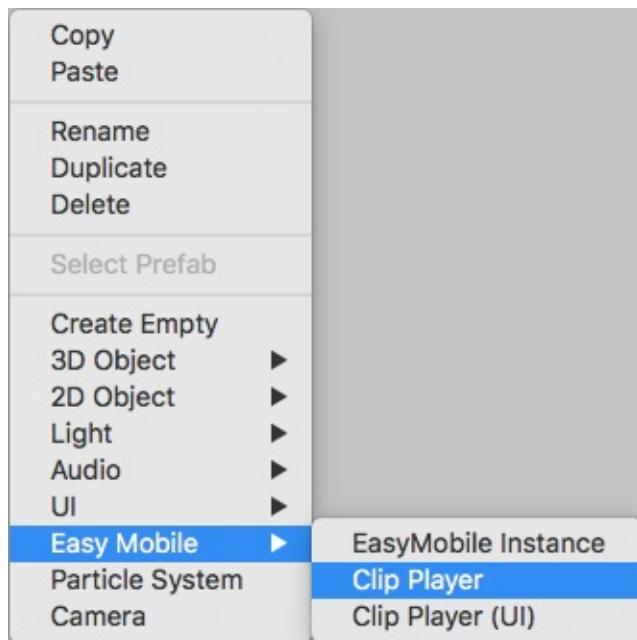
## Playback

Easy Mobile provides two built-in objects dedicated for playing recorded clips: the Clip Player and Clip Player UI objects. You can create them from the context menu (as you would with other Unity built-in objects), configure them in the inspector, and start or stop their

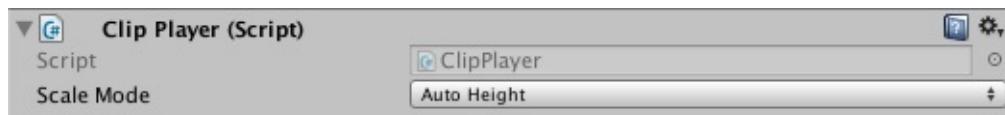
playing activity from script (see the **Scripting** section).

## Clip Player

The Clip Player is a non-UI object, which is basically a Quad object equipped with a ClipPlayer component. It is meant to be used inside the game world. To create a Clip Player object, right-click in the Hierarchy window to open the context menu, then select *Easy Mobile > Clip Player*.



Each Clip Player object contains a ClipPlayer component.

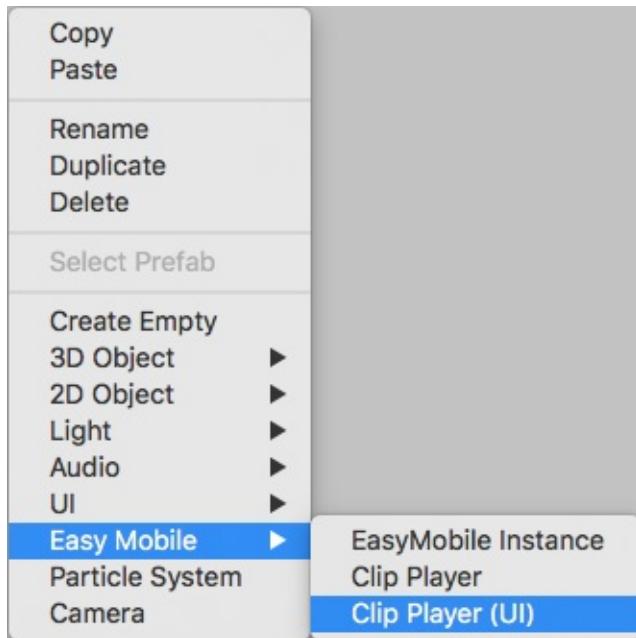


The only parameter of this component is the *Scale Mode*, which can take one of 3 values:

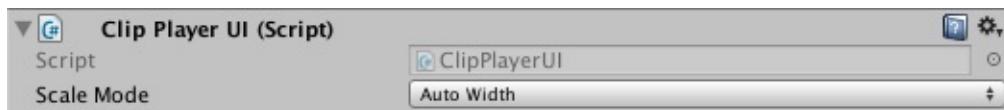
- *None*: don't adjust the object sizes
- *Auto Height*: keeps the current height of the object (the Y component of its localScale), and adjust the width (the X component of its localScale) to match the aspect ratio of the clip being played
- *Auto Width*: keeps the current width of the object (the X component of its localScale), and adjust the height (the Y component of its localScale) to match the aspect ratio of the clip being played

## Clip Player UI

The Clip Player UI, as its name implies, is a UI object living inside a Canvas. It is the object to use when you want to play a clip inside the UI. It is basically a Raw Image object equipped with a ClipPlayerUI component. To create a Clip Player UI object, right-click in the Hierarchy window to open the context menu, then select *Easy Mobile > Clip Player (UI)*.



Each Clip Player UI object contains a ClipPlayerUI component.



The only parameter of this component is the *Scale Mode*, which can take one of 3 values:

- *None*: don't adjust the object sizes
- *Auto Height*: keeps the current height of the object (the Height value in its Rect Transform), and adjust the width (the Width value in its Rect Transform) to match the aspect ratio of the clip being played
- *Auto Width*: keeps the current width of the object (the Width value in its Rect Transform), and adjust the height (the Height value in its Rect Transform) to match the aspect ratio of the clip being played

## Custom Clip Player

Beside the two built-in clip players provided by Easy Mobile, you can construct your own player to serve your specific needs. To make it consistent with other players, and compatible with Easy Mobile API, this player should contain a script implementing the *IClipPlayer* interface, which is responsible for applying the frames (*RenderTextures*) of the clip, at the required frame rate, to whatever texture-displaying component it is equipped with.



# Scripting

This section provides a guide to work with the GIF API. At this stage, it's assumed that you have setup a recorder for the camera you want to record, and created an appropriate clip player to play the recorded clip. If you're not familiar with these concepts, please review the [Setup](#) section.

You can access all the GIF API methods via the `Gif` class under the `EasyMobile` namespace. As for Giphy API, use the `Giphy` class.

## Recording

To start recording on the created recorder, use the `StartRecording` method. You can do this as soon as the game starts; the recorder only stores a few last seconds (specified by the `Length` parameter in the Recorder inspector) of the recording, and automatically discards the rest.

```
// Use the EasyMobile namespace
using EasyMobile;

...

// Drag the camera with the Recorder component to this field in the inspector
public Recorder recorder;

...

// You can start recording as soon as your game starts
// (suppose you have a method named StartGame, which is called when the game starts)
void StartGame()
{
    // Start recording!
    Gif.StartRecording(recorder);

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

To stop recording, simply call the `StopRecording` method, passing the relevant recorder. The method returns an `AnimatedClip` object, which can be played or exported into a GIF image afterward. To continue the previous example:

```

// Use the EasyMobile namespace
using EasyMobile;

...

// Drag the camera with the Recorder component to this field in the inspector
public Recorder recorder;

// The recorded clip
AnimatedClip myClip;

// You can start recording as soon as your game starts
// (suppose you have a method named StartGame, which is called when the game starts)
void StartGame()
{
    // Start recording!
    Gif.StartRecording(recorder);

    // Do other stuff...
}

...

// A suitable time to stop recording may be when the game ends (the player dies)
// (suppose you have a method named GameOver, called when the game ends)
void GameOver()
{
    // Stop recording
    myClip = Gif.StopRecording(recorder);

    // Do other stuff...
}

```

## Playback

To play a recorded clip using a pre-created clip player, use the *PlayClip* method. This method receives as argument an *IClipPlayer* interface, which is implemented by both *ClipPlayer* and *ClipPlayerUI* classes, therefore it works with both Clip Player and Clip Player UI object. The second argument is an *AnimatedClip* object. Other arguments include an optional delay time before the playing starts, and the looping mode. You can pause, resume and stop the player using the *PausePlayer*, *ResumePlayer* and *StopPlayer* methods, respectively.

To continue the previous example:

```

// Use the EasyMobile namespace
using EasyMobile;

```

```
...

// Drag the camera with the Recorder component to this field in the inspector
public Recorder recorder;

// Suppose you've created a ClipPlayerUI object (ClipPlayer will also work)
// Drag the pre-created clip player to this field in the inspector
public ClipPlayerUI clipPlayer;

// The recorded clip
AnimatedClip myClip;

// You can start recording as soon as your game starts
// (suppose you have a method named StartGame, which is called when the game starts)
void StartGame()
{
    // Start recording!
    Gif.StartRecording(recorder);

    // Do other stuff...
}

...

// A suitable time to stop recording may be when the game ends (the player dies)
// (suppose you have a method named GameOver, called when the game ends)
void GameOver()
{
    // Stop recording
    myClip = Gif.StopRecording(recorder);

    // Play the recorded clip!
    PlayMyClip();
}

...

// This method plays the recorded clip on the created player,
// with no delay before playing, and loop indefinitely.
void PlayMyClip()
{
    Gif.PlayClip(clipPlayer, myClip);
}

// This method plays the recorded clip on the created player,
// with a delay of 1 seconds before playing, and loop indefinitely,
// (you can set loop = false to play the clip only once)
void PlayMyClipWithDelay()
{
    Gif.PlayClip(clipPlayer, myClip, 1f, true);
}

// This method pauses the player.
```

```

void PausePlayer()
{
    Gif.PausePlayer(clipPlayer);
}

// This method un-pauses the player.
void UnPausePlayer()
{
    Gif.ResumePlayer(clipPlayer);
}

// This method stops the player.
void StopPlayer()
{
    Gif.StopPlayer(clipPlayer);
}

```

## GIF Export

To export the recorded clip into a GIF image, use the *ExportGif* method. In the editor, the exported GIF file will be stored right under the *Assets* folder; on mobile devices, the storage location is [Application.persistentDataPath](#). You can specify the filename and the quality of the GIF image as well as the priority of the exporting thread. The quality setting accepts values from 1 to 100 (inputs will be clamped to this range). Bigger values will result in better looking GIFs, but will take slightly longer processing time; 80 is generally a good value in terms of time-quality balance. This method has two callbacks: one is called repeatedly during the process and receives the progress value (0 to 1), the other is called when the export completes and receives the file path of the generated image. Though the GIF generation process is done in a separate thread, these callbacks are guaranteed to be called from the main thread, so you can safely access all Unity API from within them.

In the rare case that you want to control the looping mode of the exported GIF (the default is loop indefinitely), use the variant of *ExportGif* that has a *loop* parameter (note that some GIF players may ignore this setting):

- *loop < 0*: disable looping (play once)
- *loop = 0*: loop indefinitely
- *loop > 0*: loop a number of times

In the following example, we'll export a GIF image from the recorded clip returned after the recording has stopped.

```

// Use the EasyMobile namespace
using EasyMobile;

...

```

```
// Drag the camera with the Recorder component to this field in the inspector
public Recorder recorder;

// The recorded clip
AnimatedClip myClip;

// You can start recording as soon as your game starts
// (suppose you have a method named StartGame, which is called when the game starts)
void StartGame()
{
    // Start recording!
    Gif.StartRecording(recorder);

    // Do other stuff...
}

...

// A suitable time to stop recording can be when the game ends (the player dies)
// (suppose you have a method named GameOver, called when the game ends)
void GameOver()
{
    // Stop recording
    myClip = Gif.StopRecording(recorder);

    // Export GIF image from the resulted clip
    ExportMyGif();
}

...

// This method exports a GIF image from the recorded clip.
void ExportMyGif()
{
    // Parameter setup
    string filename = "myGif";      // filename, no need the ".gif" extension
    int loop = 0;                  // -1: no loop, 0: loop indefinitely, >0: loop a set
    number of times
    int quality = 80;             // 80 is a good value in terms of time-quality balan
    ce
    System.Threading.ThreadPriority tPriority = System.Threading.ThreadPriority.Normal
; // exporting thread priority

    Gif.ExportGif(myClip,
                  filename,
                  loop,
                  quality,
                  tPriority,
                  OnGifExportProgress,
                  OnGifExportCompleted);
}
```

```
// This callback is called repeatedly during the GIF exporting process.
// It receives a reference to original clip and a progress value ranging from 0 to 1.
void OnGifExportProgress(AnimatedClip clip, float progress)
{
    Debug.Log(string.Format("Export progress: {0:P0}", progress));
}

// This callback is called once the GIF exporting has completed.
// It receives a reference to the original clip and the filepath of the generated image.
void OnGifExportCompleted(AnimatedClip clip, string path)
{
    Debug.Log("A GIF image has been created at " + path);
}
```

## Disposing of AnimatedClip

Internally, each *AnimatedClip* object consists of an array of *RenderTexture*, a "native engine object" type, which is not garbage collected as normal managed types. That means these render textures won't be "destroyed" automatically when their containing clip is garbage collected (the clip object does get collected, but the render textures it references don't, thus creating memory leaks). To take care of this issue, we have the *AnimatedClip* implement the *IDisposable* interface and provide the *Dispose* method to release the render textures, as Unity recommended. You *must* call this *Dispose* method, preferably as soon as you're done with using a clip (after playing or exporting GIF), to make sure the render textures are properly released and not cause memory issues.

We'll extend the *OnGifExportCompleted* callback handler of the previous example to dispose the recorded clip as soon as we've generated a GIF image from it.

```
// This callback is called once the GIF exporting has completed.
// It receives a reference to the original clip and the filepath of the generated image.
void OnGifExportCompleted(AnimatedClip clip, string path)
{
    Debug.Log("A GIF image has been created at " + path);

    // We've done using the clip, dispose it to save memory
    if (clip == myClip)
    {
        myClip.Dispose();
        myClip = null;
    }
}
```

## GIF Sharing

Now that a GIF image has been created, you may want to share it (because it's not fun otherwise, is it?). A common approach is to first upload the image to [Giphy](#), a popular GIF hosting site, and then share the returned URL to other social networks like Facebook and Twitter, using Easy Mobile's Native Sharing feature (see the *Native Sharing > Scripting* section, in particular the *ShareURL* method).

According to [Giphy API documentation](#), hosted Giphy URLs are supported and play on every major social network.

## Upload to Giphy

To upload a GIF image to Giphy, use the *Upload* method of the *Giphy* class. You can upload a local image on your device, or an image hosted online, provided that you have its URL. Before doing so, you'll need to prepare the upload content by creating a *GiphyUploadParams* struct. In this struct you'll specify either the file path of the local image, or the URL of the online image to upload. Note that if both parameters are provided, the local file path will be used over the URL. Within this struct you can also specify other optional parameters such as image tags, the source of the image (e.g. your website), or mark the image as private (only visible by you on Giphy). The *Upload* method has three callbacks: the first one is called repeatedly during the upload process, receiving a progress value (0 to 1); the second one is called once the upload has completed, receiving the URL of the uploaded image; and the last one will be called if the upload has failed, receiving the error message. All callbacks are called from the main thread.

### Giphy Beta and Production Key

The *Upload* method has two variants: one using Giphy's public beta key, and the other using your own channel username and production API key. The public beta key is meant to be used in development only. According to [Giphy Upload API documentation](#), it is "subject to rate limit constraints", and they "do not encourage live production deployments to use the public key". If you have created a Giphy channel and want to upload GIF images directly to that channel, you'll need to [request an Upload Production Key](#), then provide that key and your channel username to the *Upload* method.

We'll extend the above example, and modify the *OnGifExportCompleted* callback handler to upload the GIF image to Giphy once it is created. We'll demonstrate two cases: upload using the public beta key and upload using your own production key.

```
...  
  
// This callback is called once the GIF exporting has completed.  
// It receives a reference to the original clip and the filepath of the generated image.  
void OnGifExportCompleted(AnimatedClip clip, string path)
```

```
{  
    Debug.Log("A GIF image has been created at " + path);  
  
    // We've done using the clip, dispose it to save memory  
    if (clip == myClip)  
    {  
        myClip.Dispose();  
        myClip = null;  
    }  
  
    // The GIF image has been created, now we'll upload it to Giphy  
    // First prepare the upload content  
    var content = new GiphyUploadParams();  
    content.localImagePath = path;      // the file path of the generated GIF image  
    content.tags = "easy mobile, sglib games, unity";      // optional image tags, comma  
-delimited  
    content.sourcePostUrl = "YOUR_WEBSITE_ADDRESS";      // optional image source, e.g.  
your website  
    content.isHidden = false;      // optional hidden flag, set to true to mark the imag  
e as private  
  
    // Upload the image to Giphy using the public beta key  
    UploadToGiphyWithBetaKey(content);  
}  
  
// This method uploads a GIF image to Giphy using the public beta key,  
// no need to specify any username or API key here.  
void UploadToGiphyWithBetaKey(GiphyUploadParams content)  
{  
    Giphy.Upload(content, OnGiphyUploadProgress, OnGiphyUploadCompleted, OnGiphyUpload  
Failed);  
}  
  
// This method uploads a GIF image to your own Giphy channel,  
// using your channel username and production key.  
void UploadToGiphyWithProductionKey(GiphyUploadParams content)  
{  
    Giphy.Upload("YOUR_CHANNEL_USERNAME", "YOUR_PRODUCTION_KEY",  
                content,  
                OnGiphyUploadProgress,  
                OnGiphyUploadCompleted,  
                OnGiphyUploadFailed);  
}  
  
// This callback is called repeatedly during the uploading process.  
// It receives a progress value ranging from 0 to 1.  
void OnGiphyUploadProgress(float progress)  
{  
    Debug.Log(string.Format("Upload progress: {0:P0}", progress));  
}  
  
// This callback is called once the uploading has completed.  
// It receives the URL of the uploaded image.
```

```

void OnGiphyUploadCompleted(string url)
{
    Debug.Log("The GIF image has been uploaded successfully to Giphy at " + url);
}

// This callback is called if the upload has failed.
// It receives the error message.
void OnGiphyUploadFailed(string error)
{
    Debug.Log("Uploading to Giphy has failed with error: " + error);
}

```

## Display the Giphy Attribution Marks

To [request a Production Key](#), Giphy require you to display the "Powered by Giphy" attribution marks whenever their API is utilized in your app, and provide screenshots of your attribution placement when submitting for the key. To take care of this, we provide the static *IsUsingAPI* boolean property inside the *Giphy* class. This property will be true as long as Giphy API is in use, to let you know when to show their attribution marks. You can display the attribution logo using an Image or a Sprite object, then poll this property inside the *Update()* function, and activate or deactivate the object accordingly.

You can download Giphy's official attribution marks [here](#).

```

// Drag the object displaying the attribution marks to this field in the inspector
public GameObject attribution;

...

void Update()
{
    attribution.SetActive(Giphy.IsUsingAPI);
}

```

## Share Giphy URLs

After uploading your GIF image to Giphy and obtain its URL, you can share this URL using the *ShareURL* method of the *MobileNativeShare* class. In the example below, we'll modify the *OnGiphyUploadCompleted* callback handler of the previous example to store the returned URL into a global variable, which can be used for later sharing.

```
...  
  
// Global variable to hold the Giphy URL of the uploaded GIF  
string giphyURL;  
  
...  
  
// This callback is called once the uploading has completed.  
// It receives the URL of the uploaded image.  
void OnGiphyUploadCompleted(string url)  
{  
    Debug.Log("The GIF image has been uploaded successfully to Giphy at " + url);  
  
    // Store the URL into our global variable  
    giphyURL = url;  
}  
  
...  
  
// This method shares the URL using the native sharing utility on iOS and Android  
public void ShareGiphyURL()  
{  
    if (!string.IsNullOrEmpty(giphyURL))  
    {  
        MobileNativeShare.ShareURL(giphyURL);  
    }  
}
```

# 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

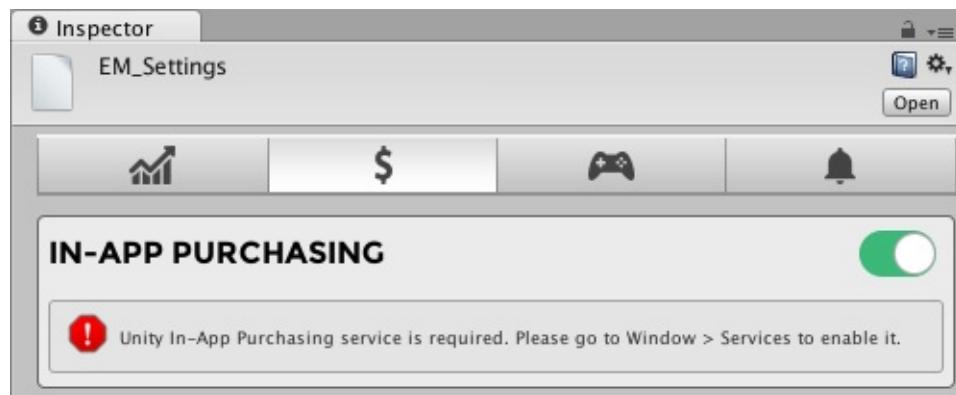
# Module Configuration

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.



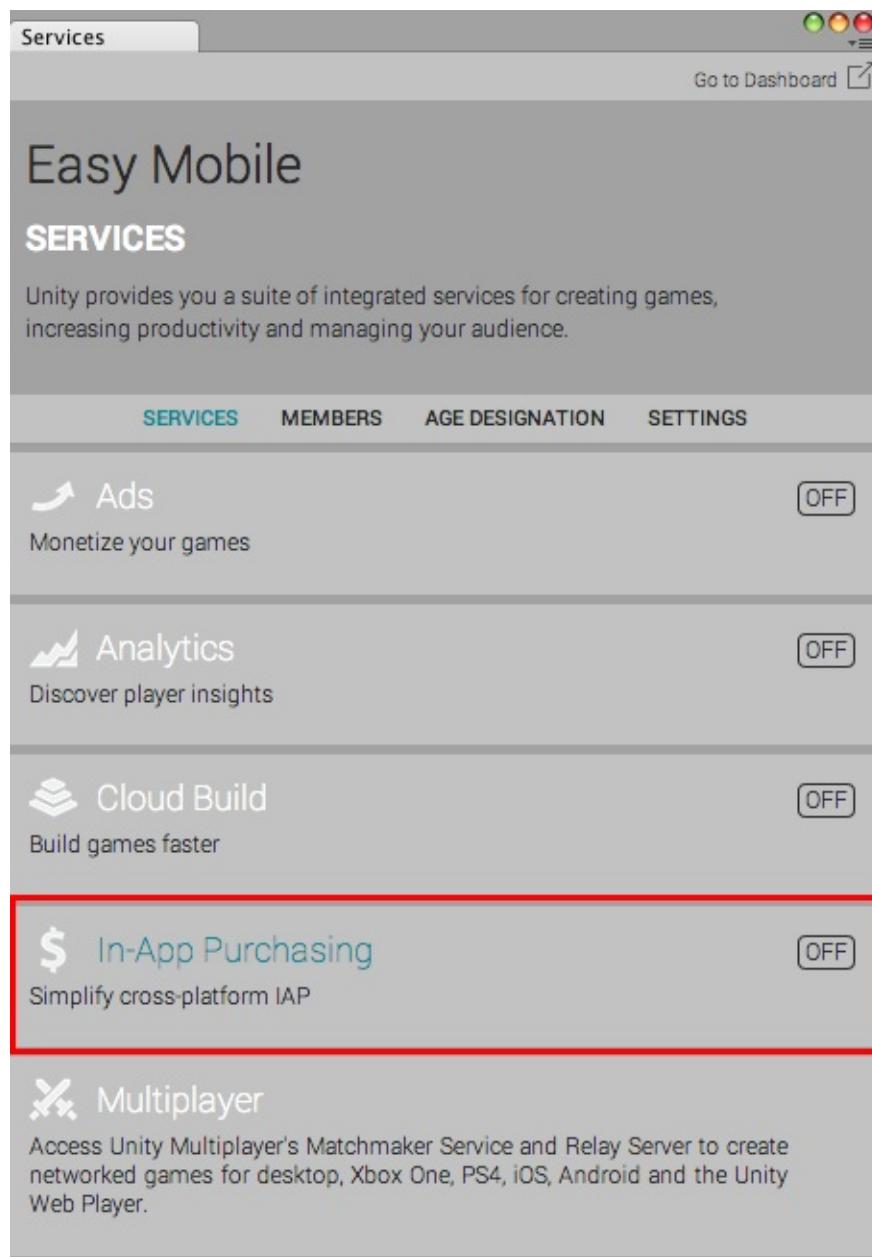
# Enable Unity IAP

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.

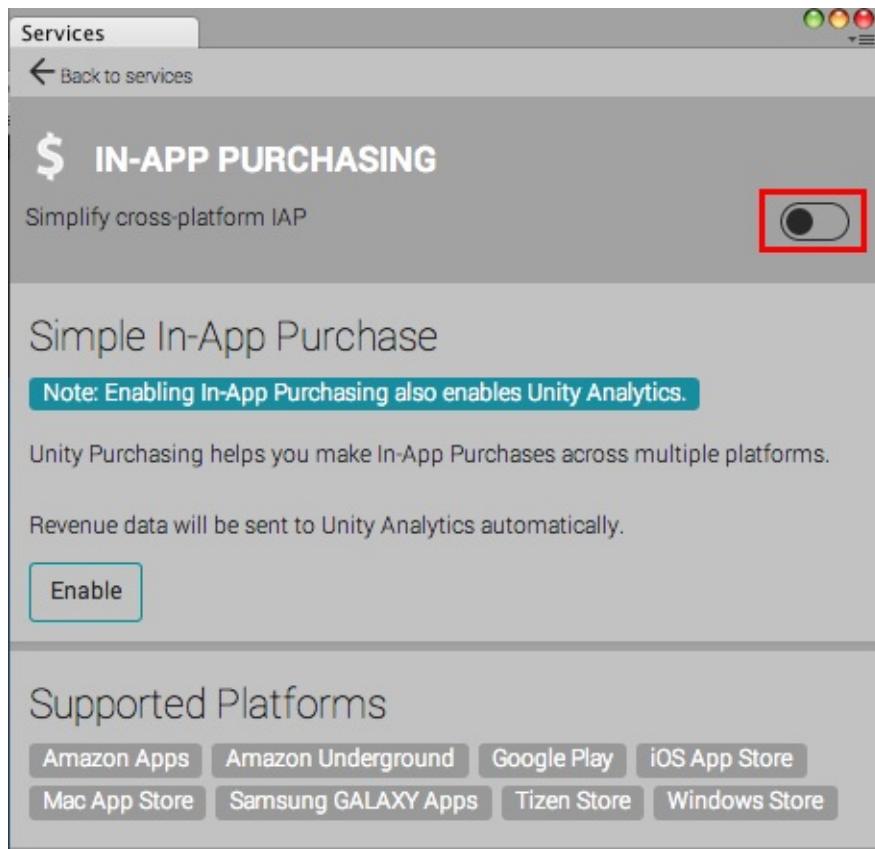


To use Unity In-App Purchasing service, you must first [set up your project for Unity Services](#).

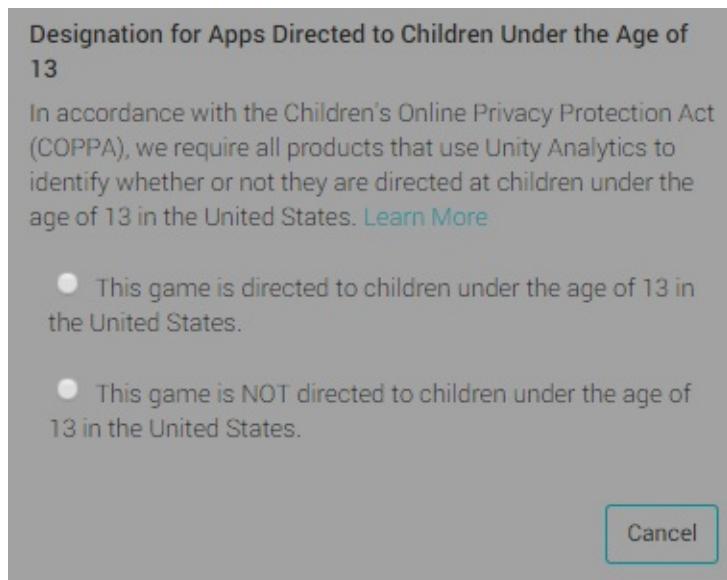
To enable Unity IAP service go to *Window > Services* and select the In-App Purchasing tab.



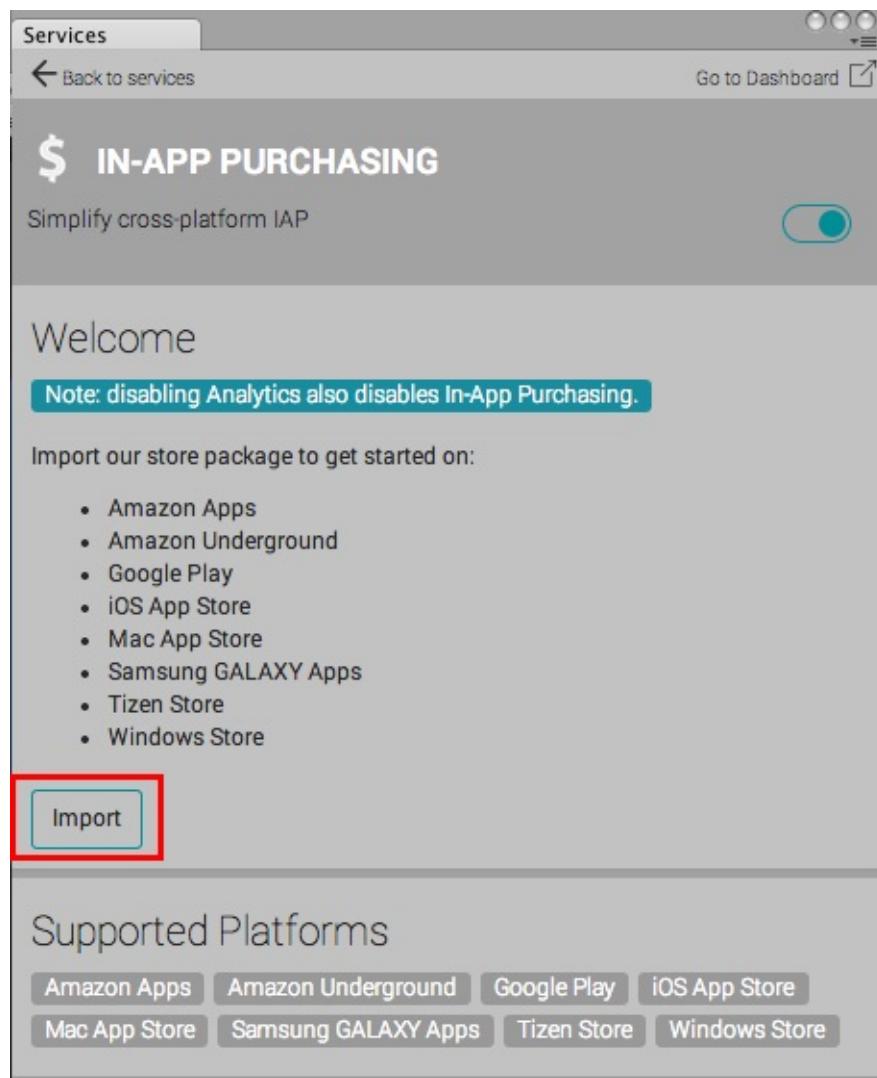
In the opened configuration window, click the toggle at the right-hand side or the *Enable* button to enable Unity IAP service.



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

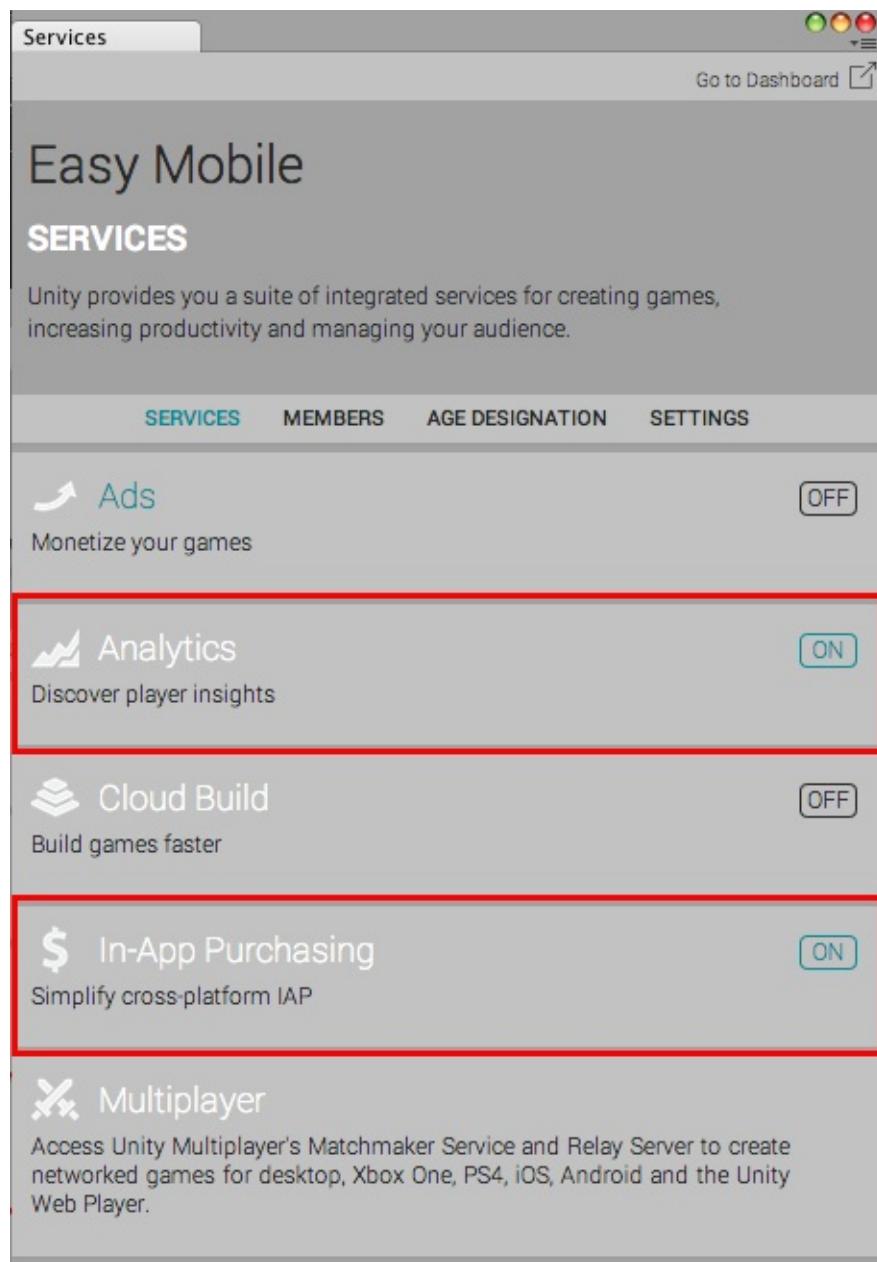


Next click the *Import* button to import the Unity IAP package to your project.

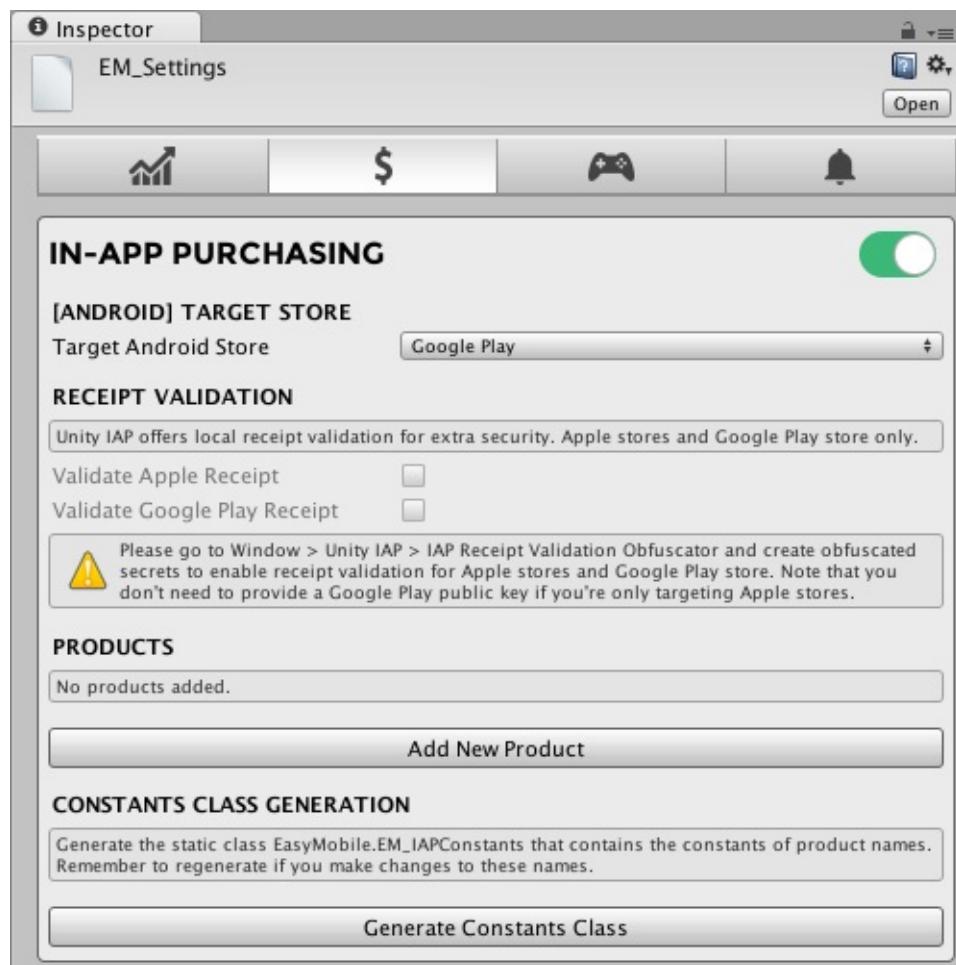


After importing, there should be a UnityPurchasing folder added under Assets/Plugins folder.

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.

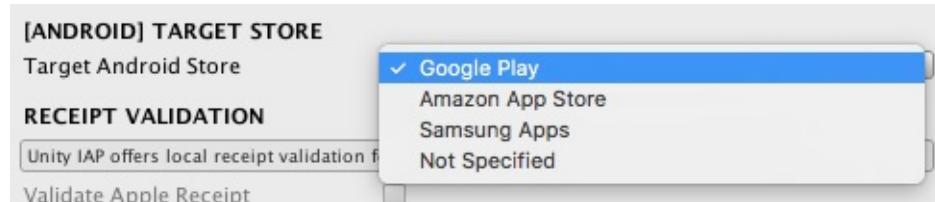


After enabling Unity IAP service, the settings interface of the In-App Purchasing module will be updated and ready for you to start configuring.



# 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.



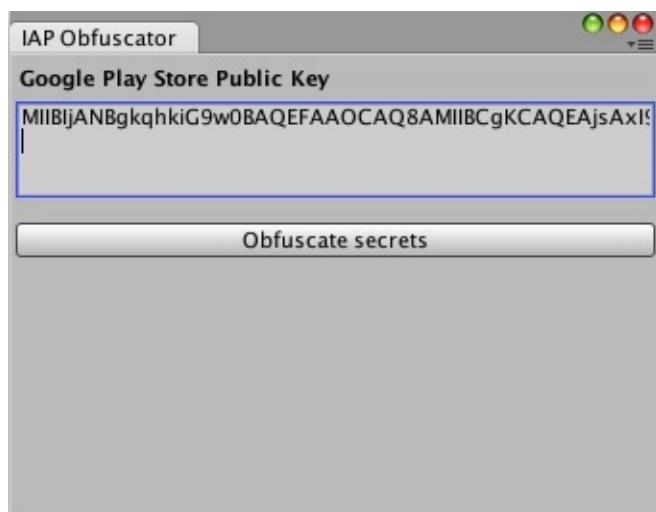
# Receipt Validation

The receipt validation feature provides extra security and helps prevent 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*.



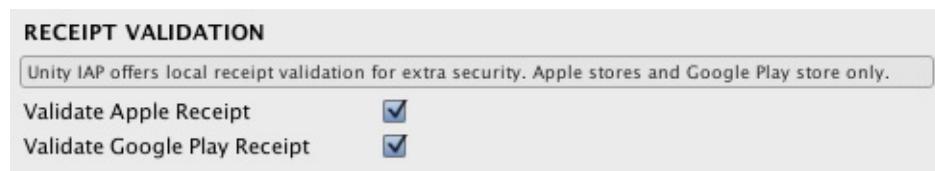
In the opened **IAP Obfuscator** window, paste in your Google Play public key and hit the *Obfuscate secrets* button. According to Unity documentation, this will obfuscate both Apple's root certificate (bundle with Unity IAP) and the provided Google Play public key and create two C# files AppleTangle 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.



# 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 light gray rectangular interface. At the top left, the word "PRODUCTS" is written in a bold, uppercase font. Below it is a horizontal button with the text "No products added." In the bottom right corner of this button is a small, faint "i" icon. At the bottom center of the interface is a large, light gray rectangular button with the text "Add New Product" in a dark gray font.

A new empty product will be added.

The screenshot shows a "New Product" dialog box. On the left, there are three input fields: "Name" (empty), "Type" (set to "Consumable"), and "Id" (empty). To the right of these fields is a vertical stack of three small buttons with arrows pointing up, down, and left respectively. At the bottom left of the dialog box is a link labeled "► More (Optional)".

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.

The screenshot shows the same "New Product" dialog box as before, but with a red rectangular box highlighting the "More (Optional)" section. This section contains two input fields: "Price" (empty) and "Description" (empty). Below these fields is a section labeled "Store-Specific Ids" with a small input field containing the value "0".

- *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 product configuration screen for a "Sample Product". The product details are as follows:

- Name:** Sample Product
- Type:** Consumable
- Id:** com.easymobile.sample\_product
- More (Optional):**
  - Price:** \$1.99
  - Description:** This is sample consumable product
- Store-Specific Ids:** A table with two rows:
 

Apple App Store	com.easymobile.sample_product_ios
Amazon Apps	com.easymobile.sample_product_amazon

On the right side of the "Store-Specific Ids" table, there are three buttons: an upward arrow, a minus sign (-), and a downward arrow. The minus sign button is highlighted with a red box.

## Remove a Product

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

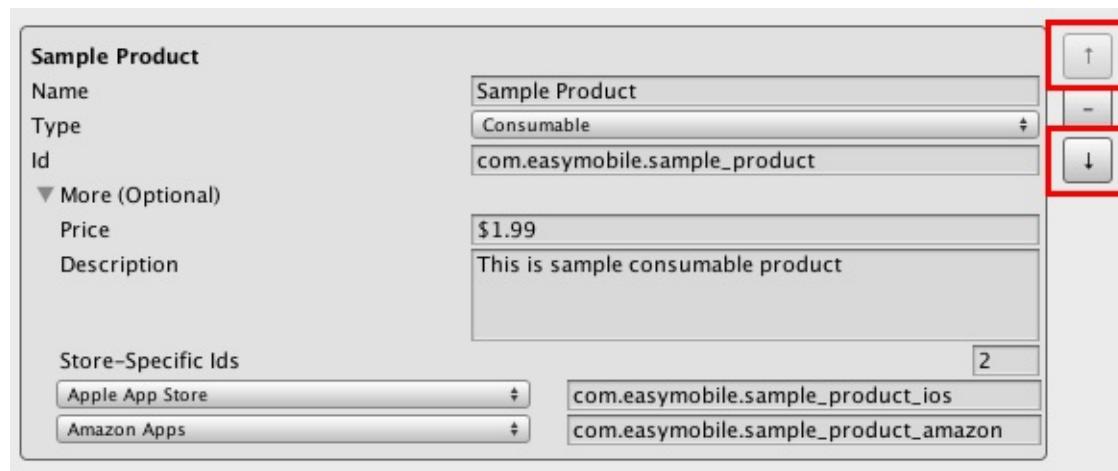
The screenshot shows the same product configuration screen as before, but the "Store-Specific Ids" table has been modified. The second row, which contained the Amazon App ID, has been removed. Only the first row (Apple App Store ID) remains.

Apple App Store	com.easymobile.sample_product_ios
-----------------	-----------------------------------

On the right side of the "Store-Specific Ids" table, the minus sign (-) button is highlighted with a red box.

## 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.



## Setup Products for Targeted Stores

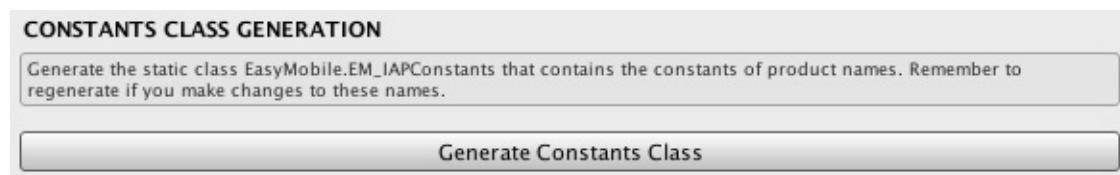
Beside creating the product list in Unity, you also need to declare similar products for your targeted stores, e.g. if you're targeting iOS App Store you need to create the products in iTunes Connect. If you're not familiar with the process, you can follow [Unity's instructions on configuring IAP for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

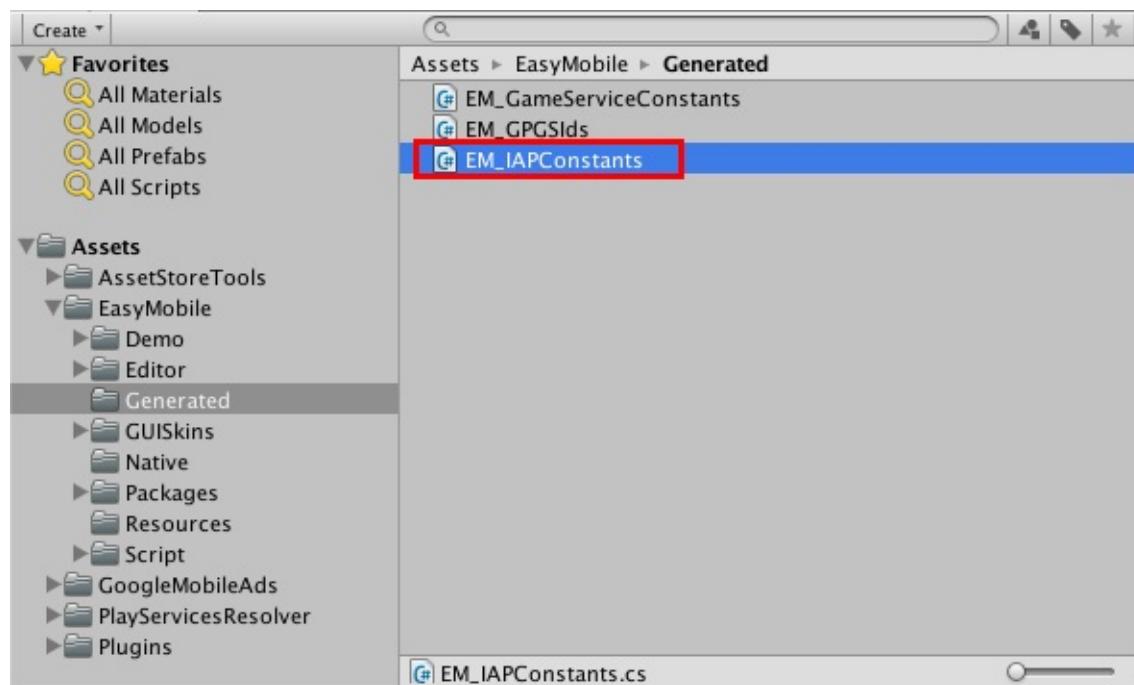
# IAP Constants Generation

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  
// IAPPProduct 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  
IAPPProduct[] products = EM_Settings.InAppPurchasing.Products;  
  
// Print all product names  
foreach (IAPPProduct 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(IAPPProduct 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(IAPPProduct 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
IAPPProduct[] products = EM_Settings.InAppPurchasing.Products;

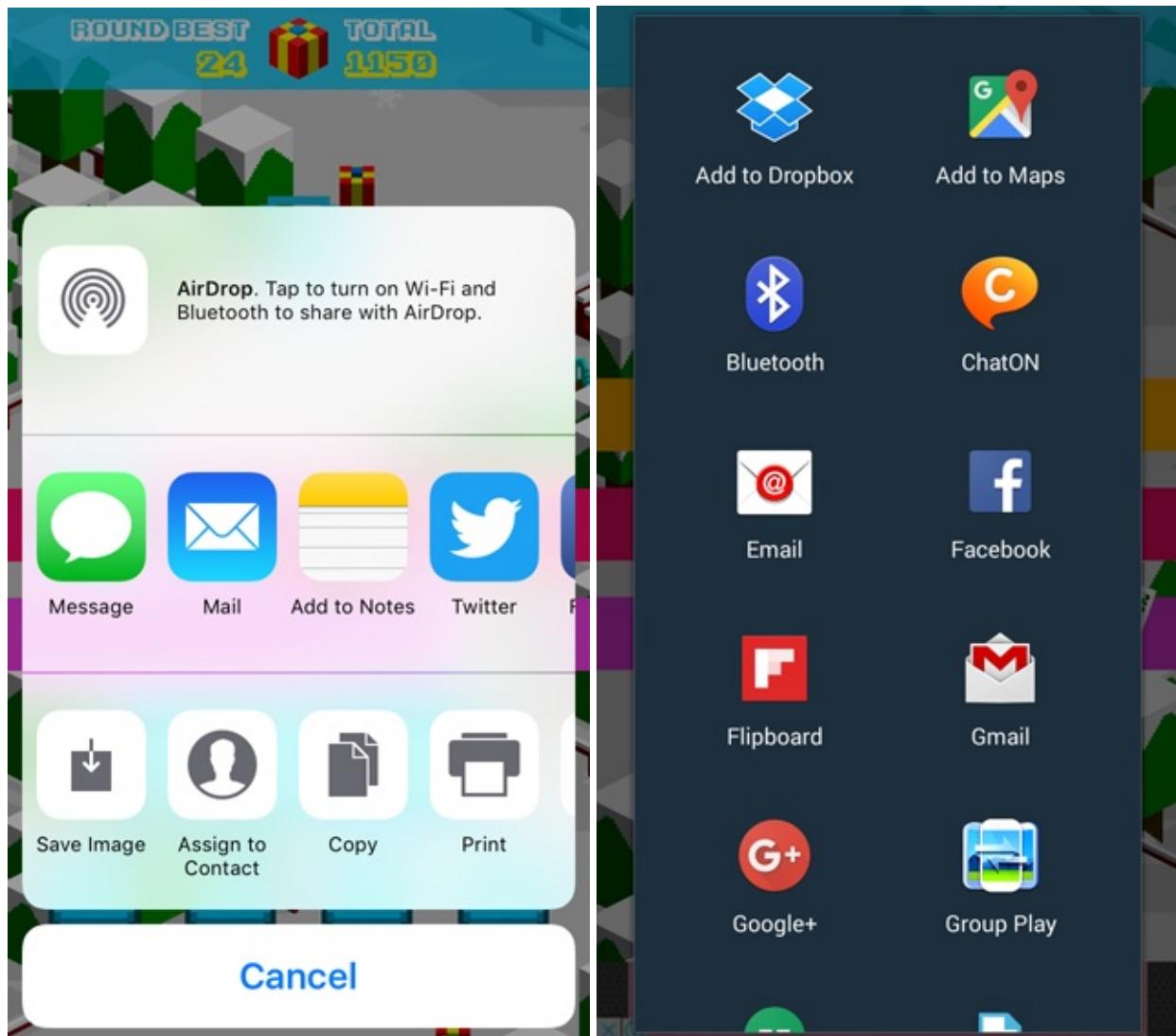
// Print products' localized data
foreach (IAPPProduct prod in products)
{
    ProductMetadata data = GetProductLocalizedData(prod.Name);

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

# 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.

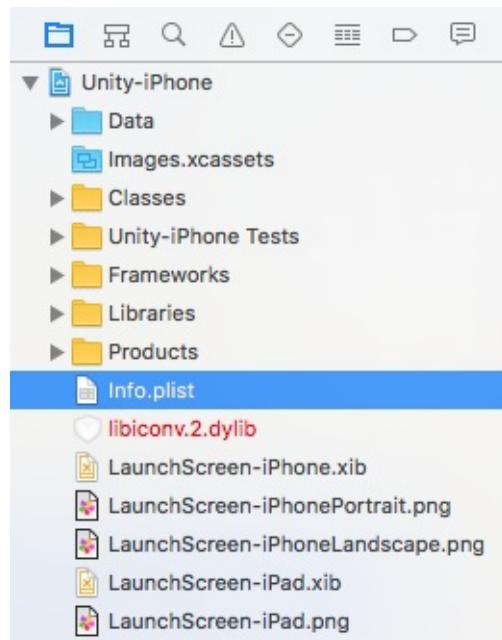
Below are the sharing interfaces on iOS and Android, respectively.



## [iOS] Request Photo Library Access Permission

Since iOS 10, in order to use the "Save Image" feature of the sharing utility, the app needs to ask for user permission before it can access the photo library. Failure to do so will cause the app to crash as soon as the user selects the option. To request the photo library access permission, you need to add the **Privacy - Photo Library Usage Description** key to the Info.plist of your Xcode project.

In your generated Xcode project open the Info.plist file.



Click the + button on the right of **Information Property List** to add a new key.

Key	Type	Value
▼ Information Property List	Dictionary	(24 items)
Localization native development region	String	en
Bundle display name	String	Bridges!
Executable file	String	\$(EXECUTABLE_NAME)
► Icon files	Array	(7 items)
Bundle identifier	String	com.sglib.\$(PRODUCT_NAME)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	0

Scroll down to find the **Privacy - Photo Library Usage Description** key.

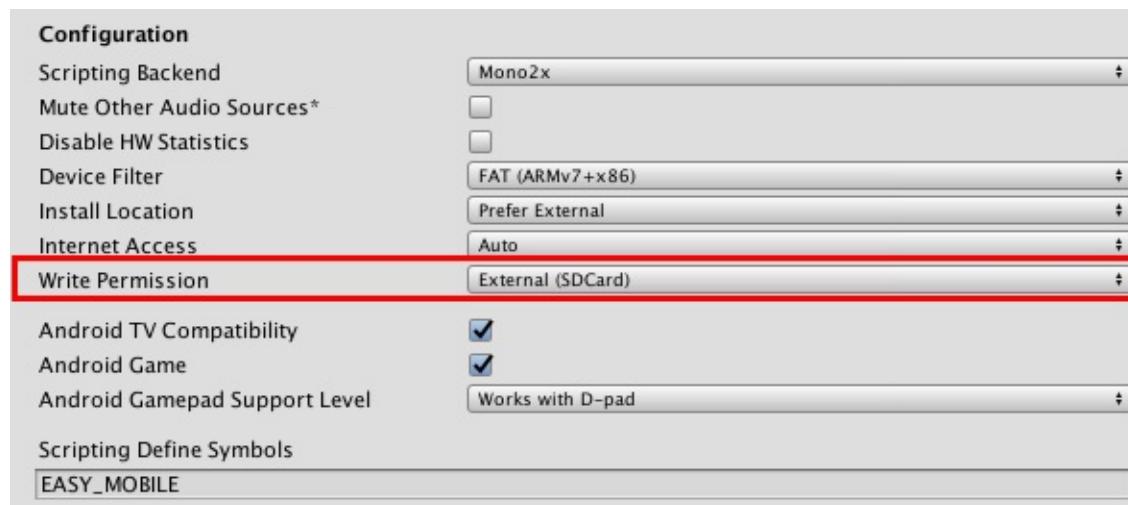
Key	Type	Value
▼ Information Property List	Dictionary	(25 items)
Privacy - Photo Library Usage Description	String	
Privacy - Reminders Usage Description	String	en
Privacy - Siri Usage Description	String	Bridges!
► Privacy - Speech Recognition Usage...	Array	(7 items)
Privacy - TV Provider Usage Descripti...	String	com.sglib.\$(PRODUCT_NAME)
Privacy - Video Subscriber Account U...	String	6.0
Quick Look needs to be run in main th...	String	\$(PRODUCT_NAME)
Quick Look preview height	String	APPL
Quick Look preview width	String	1.0
Quick Look supports concurrent requ...	String	0
Application requires iPhone environment	Boolean	YES

Enter a value for the key, this message will be displayed as the app requests access permission when the user selects the "Save Image" option.

Key	Type	Value
▼ Information Property List	Dictionary	(25 items)
Privacy - Photo Library Usage Description	String	This app wants to access the photo library
Localization native development region	String	en
Bundle display name	String	Bridges!
Executable file	String	\$(EXECUTABLE_NAME)
► Icon files	Array	(7 items)
Bundle identifier	String	com.sgilib.\$(PRODUCT_NAME)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	0

## [Android] Enable External Write Permission

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)*.



# 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 = System.IO.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 persistentDa
taPath,
// then shares it with a sample message and an empty subject
MobileNativeShare.ShareTexture2D(sampleTexture, "screenshot", "This is a sample messag
e");
```

## Share a Text

You can share a text-only message using the *ShareText* method. Note that Facebook doesn't allow pre-filled sharing messages, so the text will be discarded when sharing to this particular network.

```
// Share a text
MobileNativeShare.ShareText("Hello from Easy Mobile!");
```

## Share a URL

To share a URL, use the *ShareURL* method. On networks like Facebook or Twitter, a summary of the page will be shown if the shared URL points to a website. URLs are also useful to share GIF images hosted on sites like [Giphy](#) (see the *GIF > Scripting* section).

```
// Share a URL
MobileNativeShare.ShareURL("www.sglibgames.com");
```



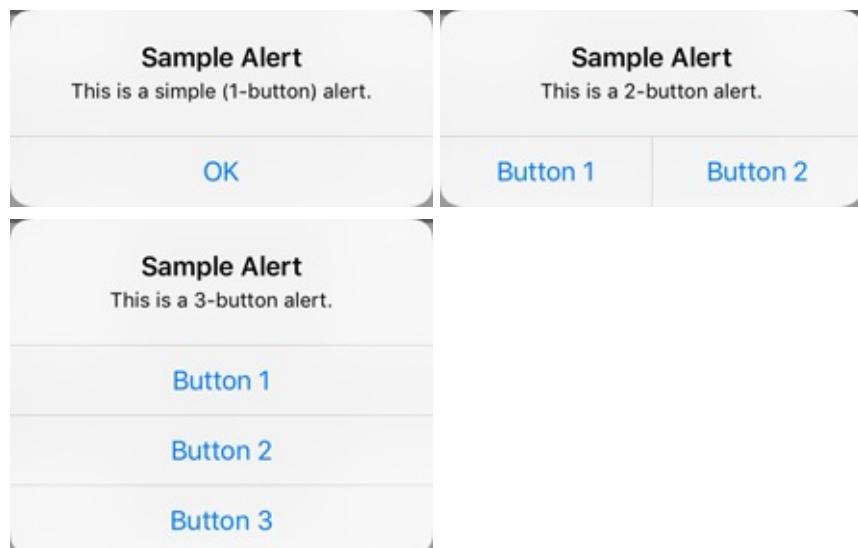
# Native UI

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.

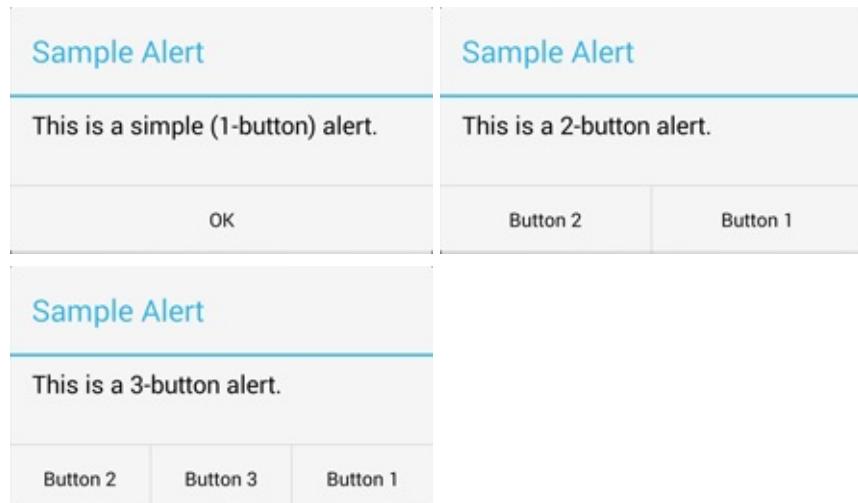
## Alerts

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.



Development Guide

# 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:

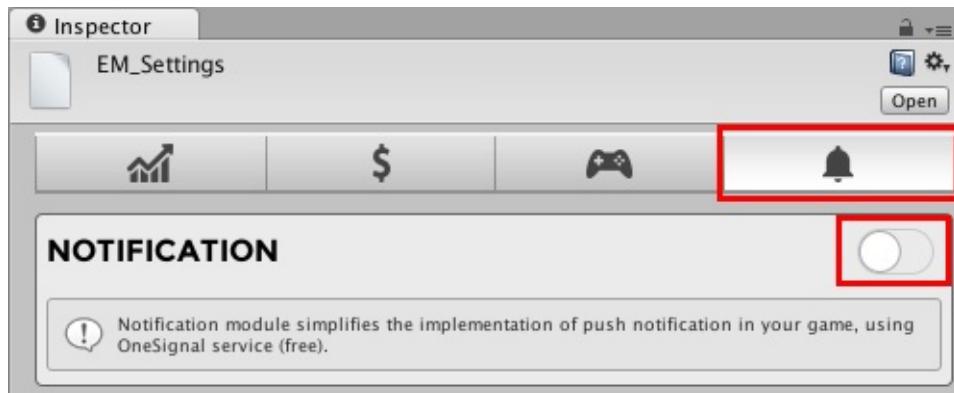
```
// Show a sample toast message
MobileNativeUI.ShowToast("This is a sample Android toast");
```

# Notification

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

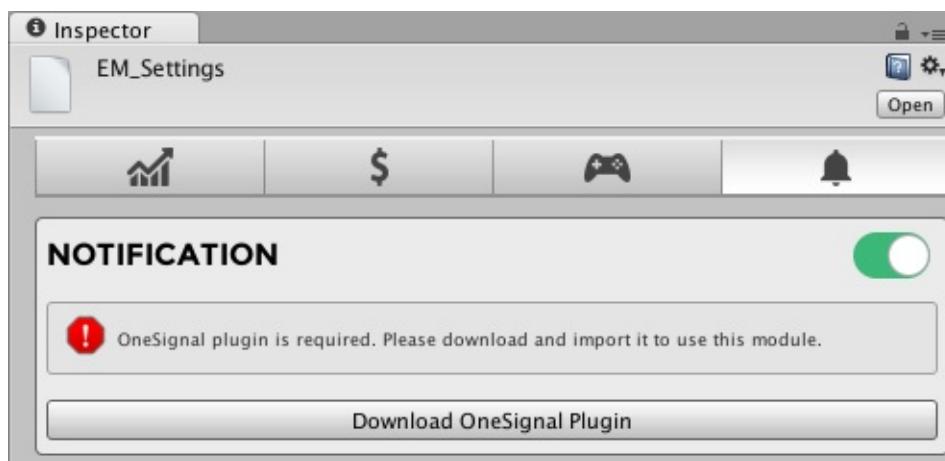
# Module Configuration

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.

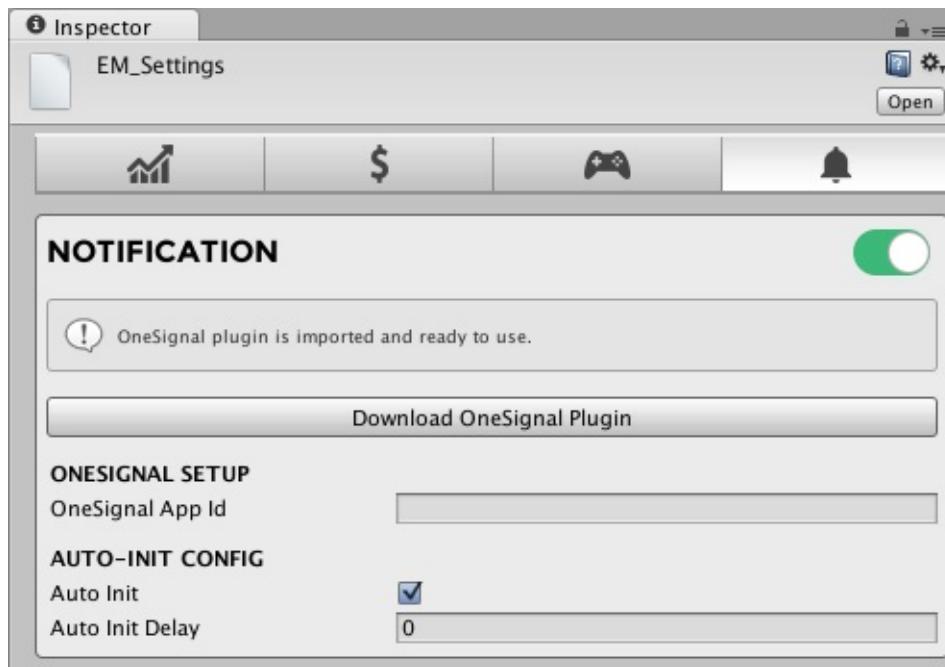


## Import OneSignal Plugin

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.

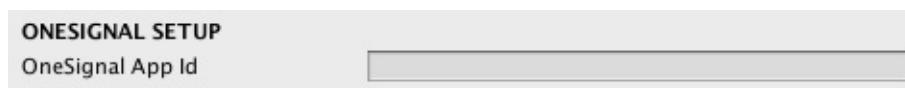


## Setup OneSignal

### 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;
}
```

# Utilities

The Utilities module is added since version 1.0.2 of Easy Mobile as a place to hold useful miscellaneous features. The first feature added to this module is Rating Request.

# Rating Request

Ratings and reviews can have a crucial impact on the performance of an app on app stores. Therefore it's a common practice to ask users for ratings when appropriate. This feature gives you an efficient way to do that using a native and highly customizable rating dialog.

This rating dialog has different appearances and behaviors depended on the platform it is being used.

## iOS 10.3 and newer

On iOS 10.3 or newer, the system-provided rating dialog is employed. This dialog is built-in to iOS since its 10.3 release, and is the preferred method to solicit user ratings on this platform.

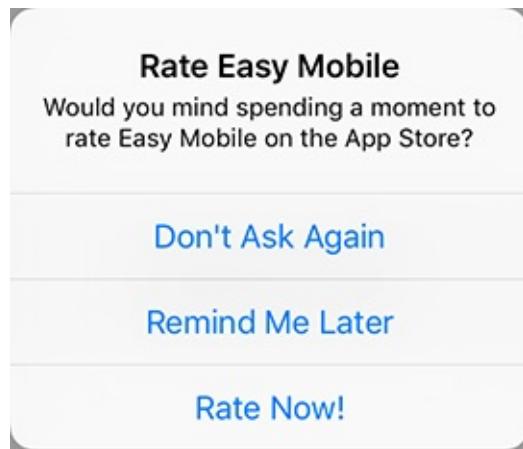
You can find more information about this built-in rating prompt at

<https://developer.apple.com/ios/human-interface-guidelines/interaction/ratings-and-reviews/>



## iOS Before 10.3

On iOS older than 10.3, a typical 3-button alert is used as the rating prompt. This is mainly for backward-compatibility purpose, since the new built-in rating prompt is preferred and will be used on the majority of iOS devices in the near future, given the high adoption rate of new iOS versions.



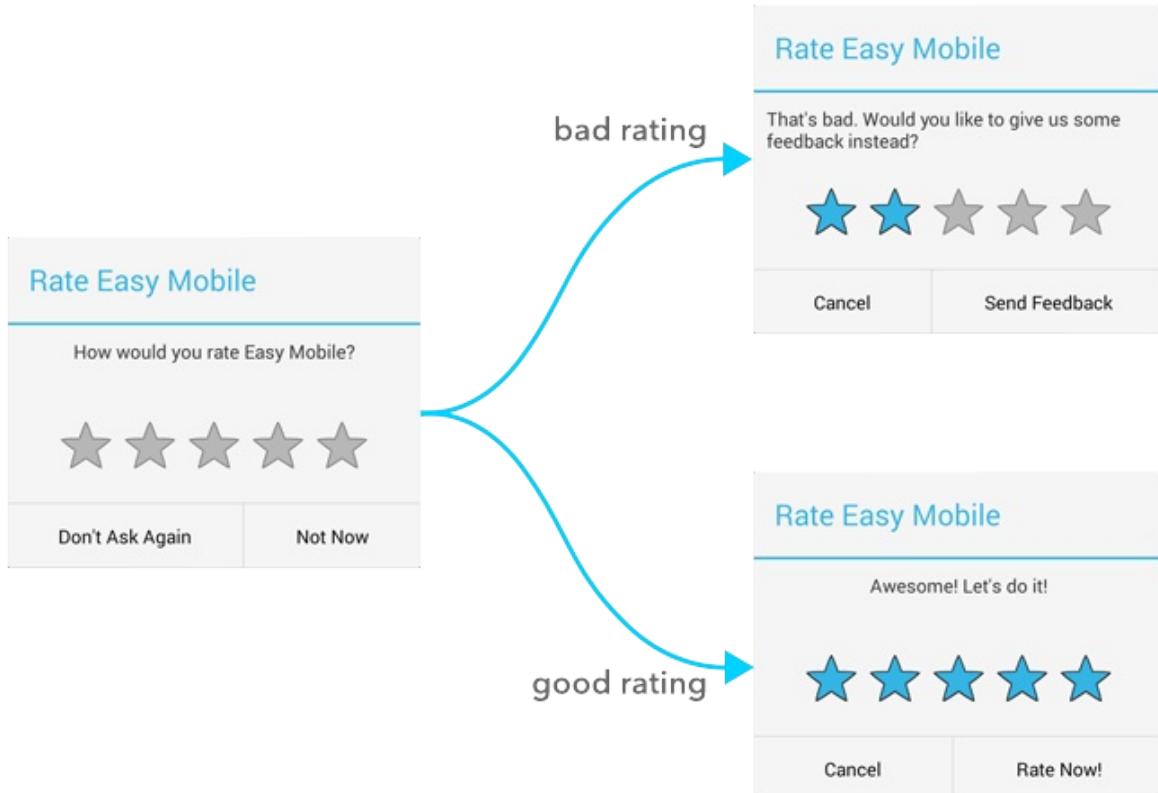
Unlike the built-in dialog, you can customize the title, message and button labels of this alert to suit your needs. The default behavior of this rating prompt is described below.

- *Don't Ask Again*: close and never show this prompt again
- *Remind Me Later*: close this alert
- *Rate Now!*: open the "Write A Review" page of the current app on the App Store, the prompt will never be displayed again

You can discard the default behavior and give your own behavior implementation if you wish.

## Android

On Android, we built a native, custom alert that employs the RatingBar component to form the rating dialog. The picture below illustrates how this dialog looks and behaves.



The idea is to ask the user how they would rate the app, and the dialog will update itself based on the given rating. You can set a "minimum accepted rating" value, which is the lowest number of stars expected for your app. Any rating lower than this value is considered a bad rating, and vice versa. If the user is giving a good rating, we will take them to the store to do the actual rating and review. Otherwise, we will suggest them to send a feedback to your support email instead. The default behavior of this rating dialog is described below. Again, you can discard this default behavior and implement a custom one if you wish.

- *Don't Ask Again*: close and never show this prompt again
- *Not Now/Cancel*: close this prompt
- *Send Feedback*: open email client for the user to send feedback to your email address
- *Rate Now!*: open the product page of the app on the Google Play Store, the prompt will never be displayed again

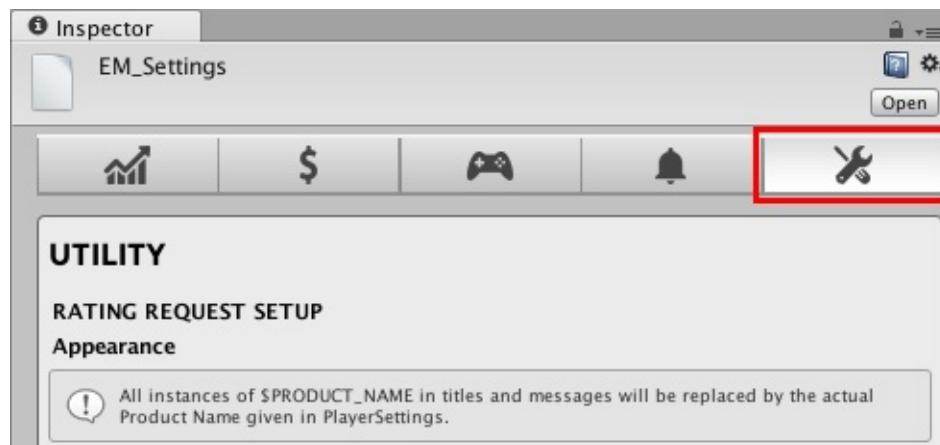
## Display Policy

It is up to you to show the rating prompt in your game whenever you see appropriate. However, it is advisable to not annoy the user by asking repeatedly or too frequently. For that purpose, the rating request feature provides an *Annual Cap* constraint, which limits the display of the rating prompt to a certain number of occurrences within a year. You can set this limit to a reasonable value in order to maintain a good user experience.

Note that on iOS 10.3 and newer (which offer the built-in rating prompt), this limit is governed by the OS and is always set to 3 occurrences per app within a 365-day period. On Android, and iOS older than 10.3, you can set this limit to any value between 3 and 100. However, big values may be harmful to user experience. Relatively small ones, e.g. 12, may be more appropriate.

# Configuration

You can configure the Rating Request feature from the Utility module. Go to *Window > Easy Mobile > Settings*, then select the Utility tab to reveal it.



In the **RATING REQUEST SETUP** section, you can customize the appearance and behavior of the request dialog.

 A detailed view of the 'RATING REQUEST SETUP' configuration panel. It includes sections for 'Appearance' (with a note about product name replacement), '[iOS pre-10.3] Dialog Content' (with fields for Title, Message, Postpone Button Text, Refuse Button Text, and Rate Button Text), '[Android] Dialog Content' (with fields for Title, Start Message, Low Rating Message, High Rating Message, Postpone Button Text, Refuse Button Text, Cancel Button Text, Feedback Button Text, and Rate Button Text), 'Behaviour' (with fields for Minimum Accepted Rating, Support Email, iOS App Id, and Annual Cap), and a 'Feedback' section (with a scrollable list of feedback messages).
 

Category	Setting	Value
[iOS pre-10.3] Dialog Content	Title	Rate \$PRODUCT_NAME
	Message	Would you mind spending a moment to rate \$PROD
	Postpone Button Text	Remind Me Later
	Refuse Button Text	Don't Ask Again
	Rate Button Text	Rate Now!
[Android] Dialog Content	Title	Rate \$PRODUCT_NAME
	Start Message	How would you rate \$PRODUCT_NAME?
	Low Rating Message	That's bad. Would you like to give us some feedback?
	High Rating Message	Awesome! Let's do it!
	Postpone Button Text	Not Now
	Refuse Button Text	Don't Ask Again
	Cancel Button Text	Cancel
	Feedback Button Text	Send Feedback
	Rate Button Text	Rate Now!
	Behaviour	Minimum Accepted Rating
Support Email		
iOS App Id		
Annual Cap		100

- *[iOS pre-10.3] Dialog Content*: for customizing the title, message and button labels of the rating prompt used on iOS versions older than 10.3

- *[Android] Dialog Content*: for customizing the title, messages and button labels of the rating prompt used on Android
- *Minimum Accepted Rating*: the lowest number of stars required to be considered as a good rating, you can set it to 0 to disable the feedback feature (accept all ratings); note that this is only applicable on Android
- *Support Email*: your email address for receiving feedback
- *iOS App Id*: your app Id on the Apple App Store, this is required to open the review page of the app on iOS older than 10.3
- *Annual Cap*: the maximum number of rating requests allowed each year; on iOS 10.3 or newer, this is always set to 3 by the system

# Scripting

This section provides a guide to work with the Request Rating API.

You can access all the Request Rating API methods via the `MobileNativeRatingRequest` class under the `EasyMobile` namespace.

## Request Rating

To show the rating dialog and retain its default behavior, use the `RequestRating` method without any parameter. Note that this method is a no-op if the rating dialog has been disabled, or the annual cap has been reached. You should call this method when it makes sense in the experience flow of your app, to maximize the effectiveness of the request.

On iOS 10.3 or newer, the actual display of the rating dialog is governed by App Store policy. When your app is still in development mode, the dialog is always displayed for testing purpose. However, it won't be shown in an app that you distribute using TestFlight.

```
// Show the rating dialog with default behavior
MobileNativeRatingRequest.RequestRating();
```

To check if the rating dialog has been disabled (because the user selected *Don't Ask Again* or already gave a rating):

```
// Check if the rating dialog has been disabled
bool isEnabled = MobileNativeRatingRequest.IsRatingRequestDisabled();
```

To get the number of remaining requests in the current year:

```
// Get the number of unused requests this year
int unusedRequests = MobileNativeRatingRequest.GetThisYearRemainingRequests();
```

To check if it's eligible to show the rating dialog (which means it hasn't been disabled and the annual cap hasn't been reached):

```
// Check if it's eligible to show the rating dialog and then show it
if (MobileNativeRatingRequest.CanRequestRating())
{
    MobileNativeRatingRequest.RequestRating();
}
```

## Request Rating with Custom Behavior

You can discard the default behavior of the rating dialog and provide your own implementation to suit your needs. This can be useful in cases when you want to perform additional tasks like recording the number of users who gave good ratings (maybe for analytics purpose). To do so, simply call the *RequestRating* method passing a callback where you implement the custom behavior. This callback takes as input an enum value of the user action, which you can use to decide whatever action should be taken. Note that you can use the *DisableRatingRequest* method to prevent the rating dialog from being displayed in the future, if the user selects "Don't Ask Again" option.

From the analytics point of view, it's worth noting that the rating given in the rating dialog on Android is merely *a suggestion of how the user would rate the app*. There's currently no reliable way to verify if it is the actual rating given on the app stores or not.

```
// Show rating dialog with a callback for custom behavior
MobileNativeRatingRequest.RequestRating(RatingCallback);

// The rating callback
private void RatingCallback(MobileNativeRatingRequest.UserAction action)
{
    switch (action)
    {
        case MobileNativeRatingRequest.UserAction.Refuse:
            // Don't ask again. Disable the rating dialog
            // to prevent it from being shown in the future.
            MobileNativeRatingRequest.DisableRatingRequest();
            break;
        case MobileNativeRatingRequest.UserAction.Postpone:
            // User selects Not Now/Cancel button.
            // The dialog automatically closes.
            break;
        case MobileNativeRatingRequest.UserAction.Feedback:
            // Bad rating, user opts to send feedback email.
            break;
        case MobileNativeRatingRequest.UserAction.Rate:
            // Good rating, user wants to rate.
            break;
    }
}
```



# Release Notes

## Version 1.1.0

This is a major release with many new features and improvements!

- **Introduces brand new module **GIF**!**
  - Low overhead screen/camera recorder
  - Built-in players for playback of recorded clips
  - High performance, mobile-friendly GIF image generator
  - Giphy upload API for sharing GIF images to social networks
- **Native Sharing module:**
  - Adds `ShareText` and `ShareURL` methods to `MobileNativeShare` class
- **Editor:**
  - Adds a new context menu for creating EasyMobile instance and other built-in objects in the Hierarchy window
  - Adds new item **Reimport Native Package** to Easy Mobile menu
  - [Unity 5.6+] Displays a warning popup when an iOS or Android build starts if no EasyMobile instance was added to any scene

## Version 1.0.4

Adds `SignOut` method to `GameServiceManager` class.

## Version 1.0.3

This update introduces important improvements and bug fixes.

- **Advertising module:**
  - AdMob rewarded ad is now supported
  - New ad network support: **AdColony**
  - Ad events are now raised from main thread when using AdMob
  - `RewardedAdCompleted` event is now raised after the ad is closed, to ensure a consistent behavior across different ad networks
- **Native Sharing module:**

- Fixed a potential memory leak issue caused by the `SaveScreenshot` method of the `MobileNativeShare` class

## Version 1.0.2

- Introduces the whole new module **Utilities**, which comes with the **Rating Request** feature, an effective way to ask for rating using a native and highly customizable "rate my app" popup.
- Updates `GameServiceManager` class, introducing new events `UserLoginSucceeded` and `UserLoginFailed`; event `UserAuthenticated` is now obsolete.

## Version 1.0.1

Updates **Game Service** module to be compatible with version 0.9.37 of the Google Play Games plugin for Unity.

## Version 1.0.0

First release.

# Upgrade Guide

This section describes the required actions you may need to take when upgrading to a certain version. Please visit this place before upgrading Easy Mobile to avoid unnecessary issues.

## Upgrading to version 1.1.0

If you're upgrading from an older version to version 1.1.0, you'll need to:

1. Remove the EasyMobile/Demo folder
2. Remove the EasyMobile/Script folder
3. Import the new version