sglib

EASY MOBILE *LITE*

User Guide

Table of Contents

Easy Mobile Lite User Guide

This document is the official user guide for Easy Mobile Lite, the stripped down version of Easy Mobile, a Unity plugin by SgLib Games.

Important Links

- Get the Full Version
- Online Documentation
- Demo APK
- Easy Mobile Lite on Unity Asset Store

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 Lite is the stripped down version of Easy Mobile and contains two modules:

- **Notification**
    - Compatible with OneSignal, a free and popular service for push notifications
- **Native Sharing**
    - Shares texts and images to social networks using the native sharing functionality

Requirements

- Unity 5.3.0 or above.

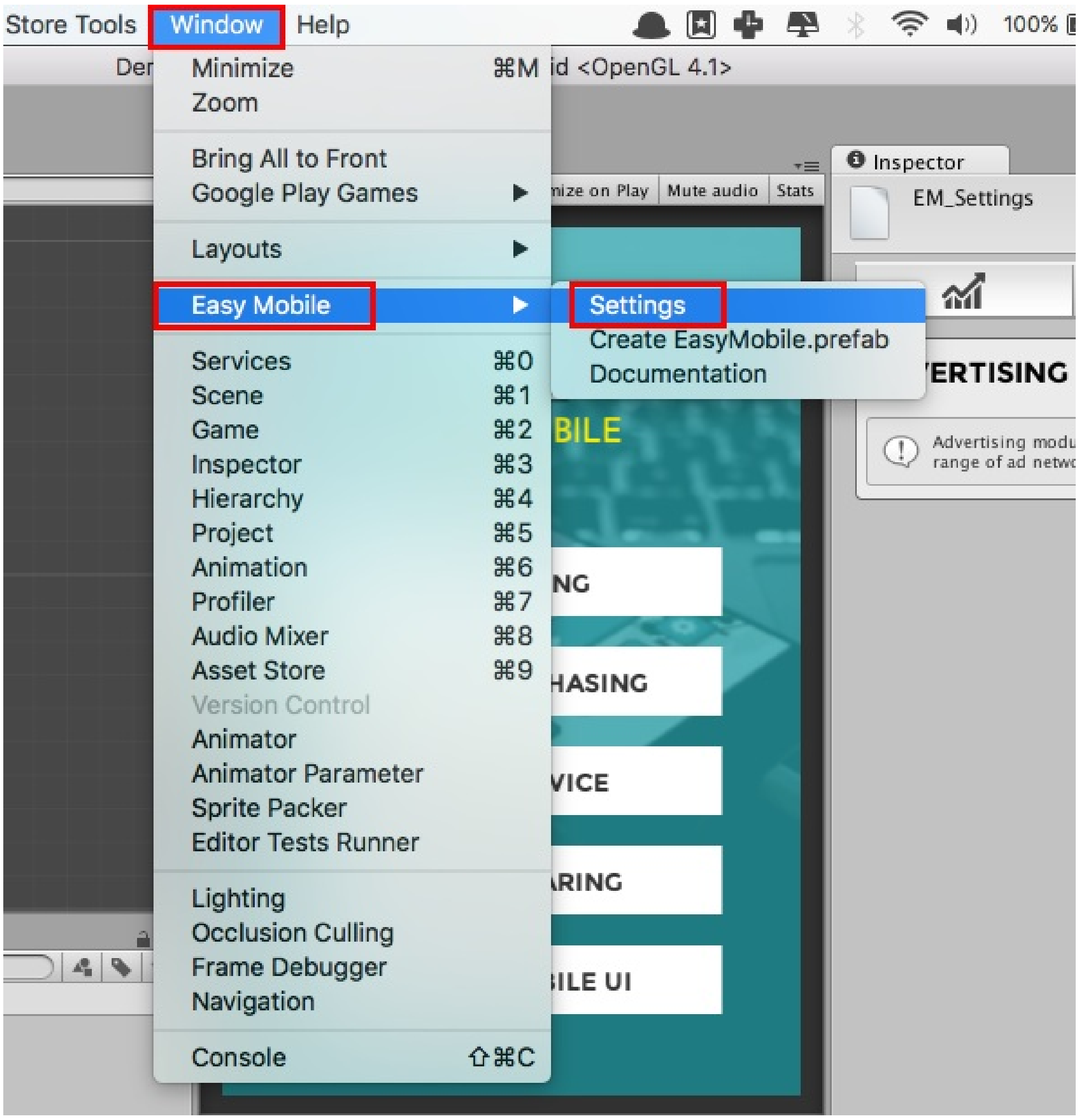
Using Easy Mobile

Using Easy Mobile involves 3 tasks:

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

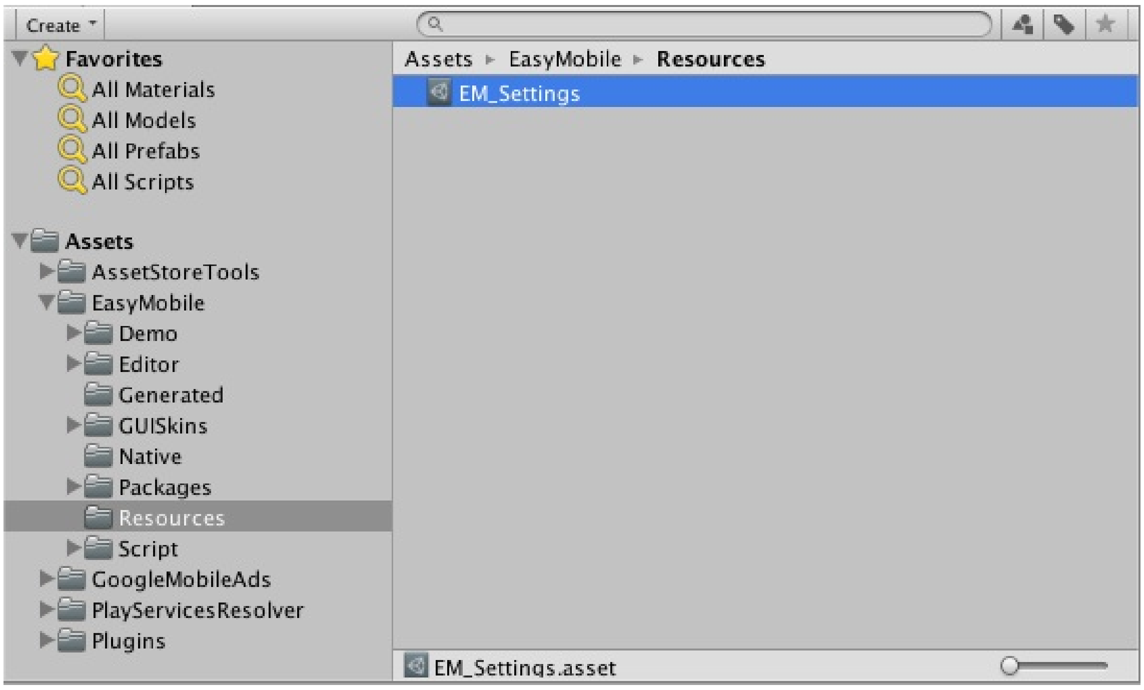
Configuration

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



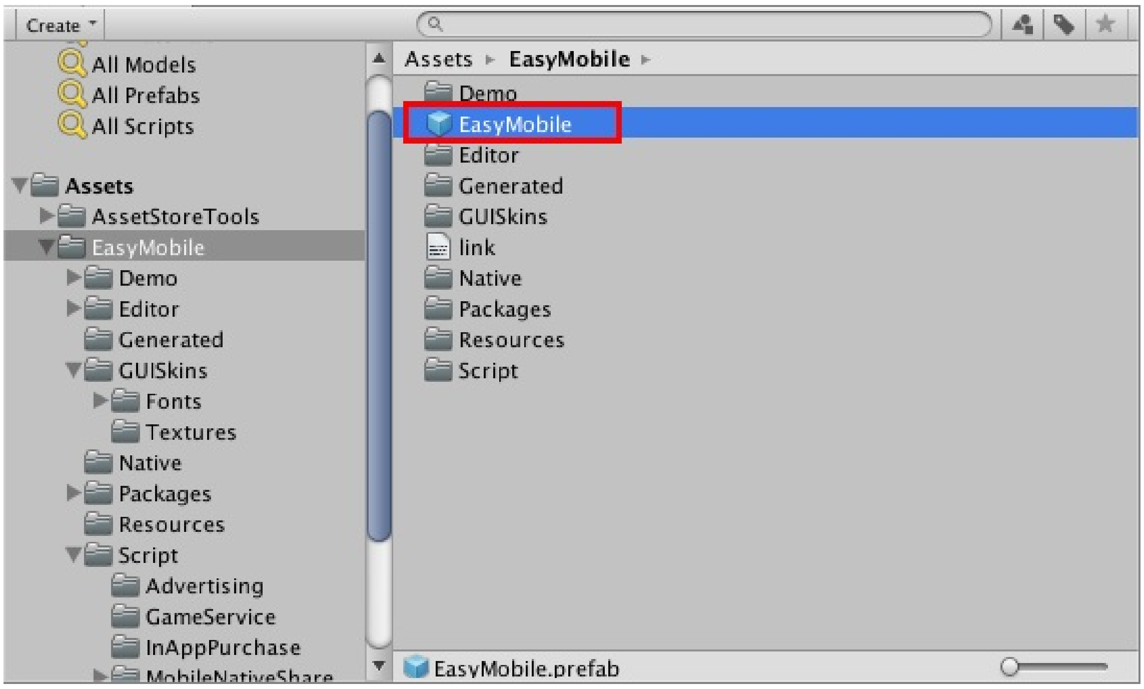
The Settings interface is the only place you go to configure the plugin. Here you can enable or disable modules, provide ads credentials, add leaderboards, create a product catalog, etc.

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



EasyMobile Prefab

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



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

Scripting

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

```
using EasyMobile;
```

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

Testing Using the Demo App

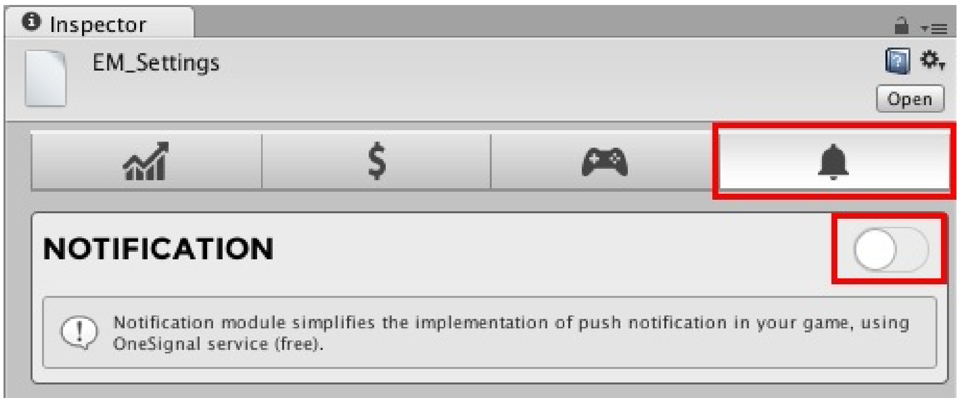
Easy Mobile comes with a demo app that you can use to quickly test each module' s operation after configuring. The demo app is contained in folder *Assets/EasyMobile/Demo*. To use the demo app, you need to add the EasyMobile prefab to the demo scene before building it.

Notification

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

Module Configuration

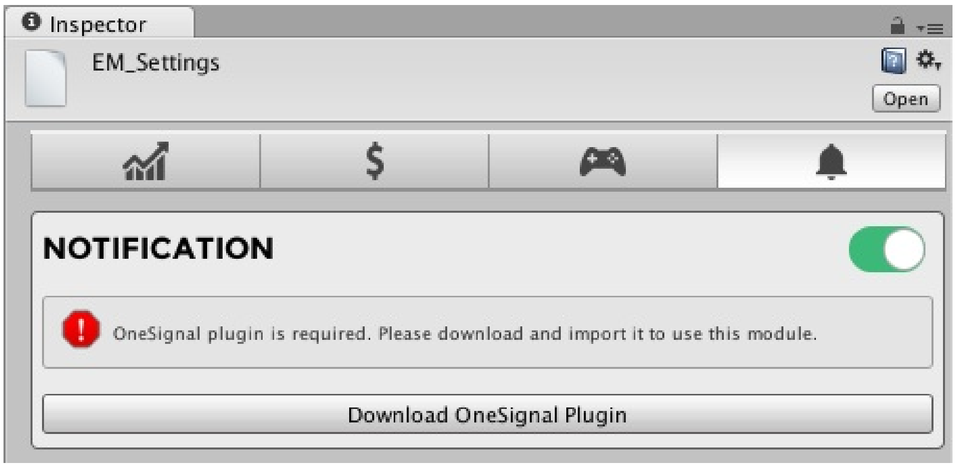
To use the Notification module you must first enable it. Go to *Window* > *Easy Mobile* > *Settings*, select the Notification tab, then click the right-hand side toggle to enable and start configuring the module.



Notification module simplifies the implementation of push notification in your game, using OneSignal service (free).

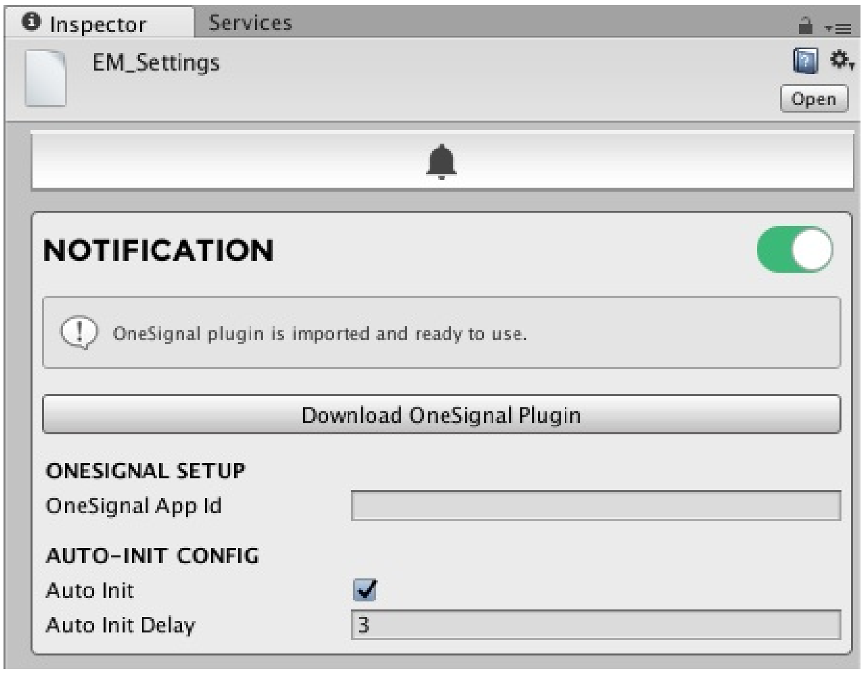
Import OneSignal Plugin

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.



OneSignal plugin is required. Please download and import it to use this module.

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.



OneSignal plugin is imported and ready to use.

Download OneSignal Plugin

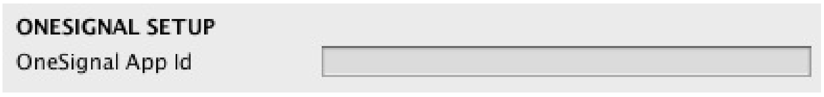
ONESIGNAL SETUP
OneSignal App Id

AUTO-INIT CONFIG
Auto Init ✓
Auto Init Delay 3

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.

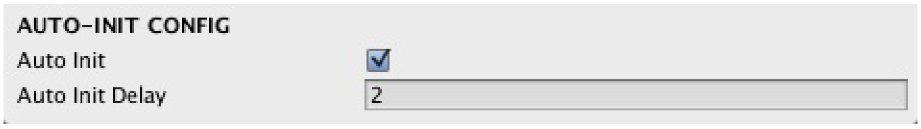
In the **ONESIGNAL SETUP** section, enter your OneSignal App ID.


ONESIGNAL SETUP
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 CONFIG
Auto Init ✓
Auto Init Delay 2

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

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

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

Scripting

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

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

Initialization

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

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

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

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

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

The *NotificationOpened* Event

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

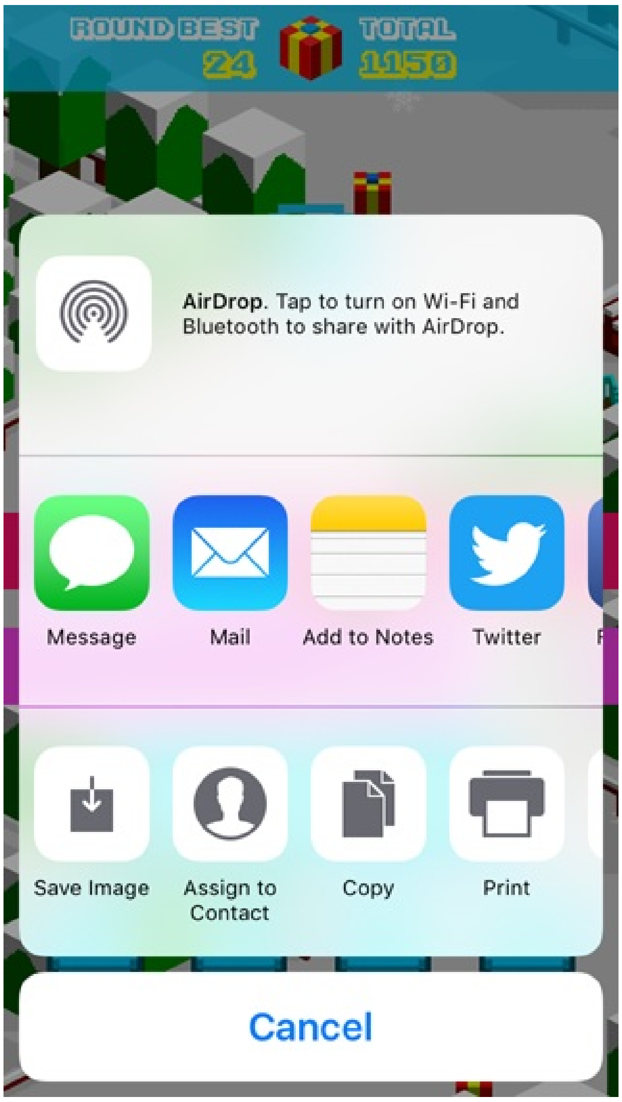
> You should subscribe to this event as early as possible, preferably as soon as your app is loaded, e.g. in the OnEnable method of a MonoBehaviour script in your first scene.

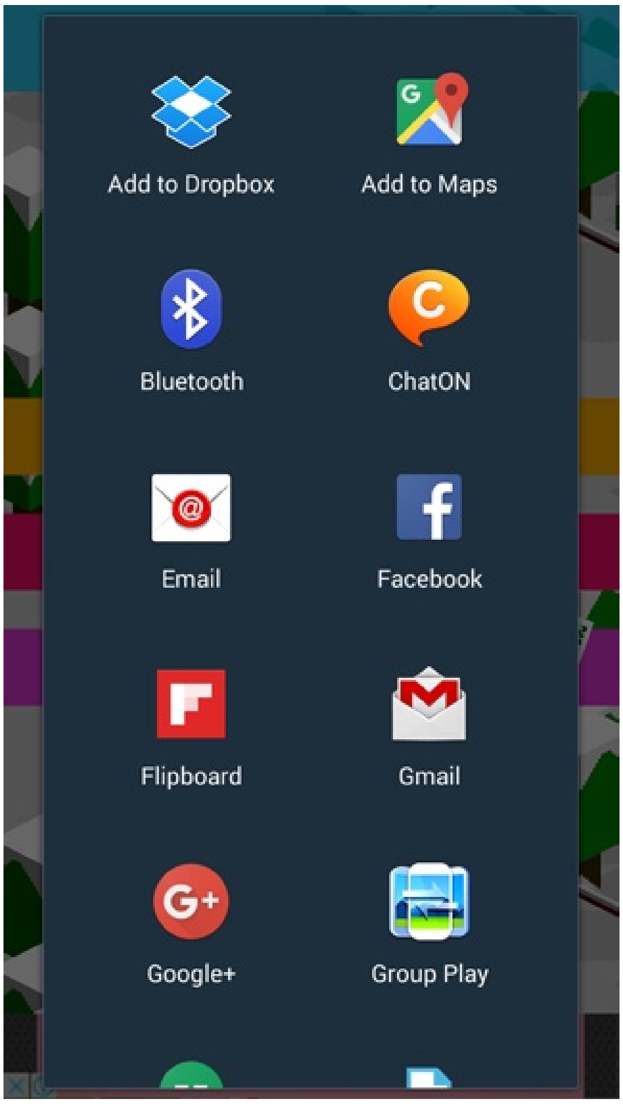
```
// Subscribe to the event
void OnEnable()
{
     NotificationManager.NotificationOpened += OnNotificationOpened;
}

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

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

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

Native Sharing

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

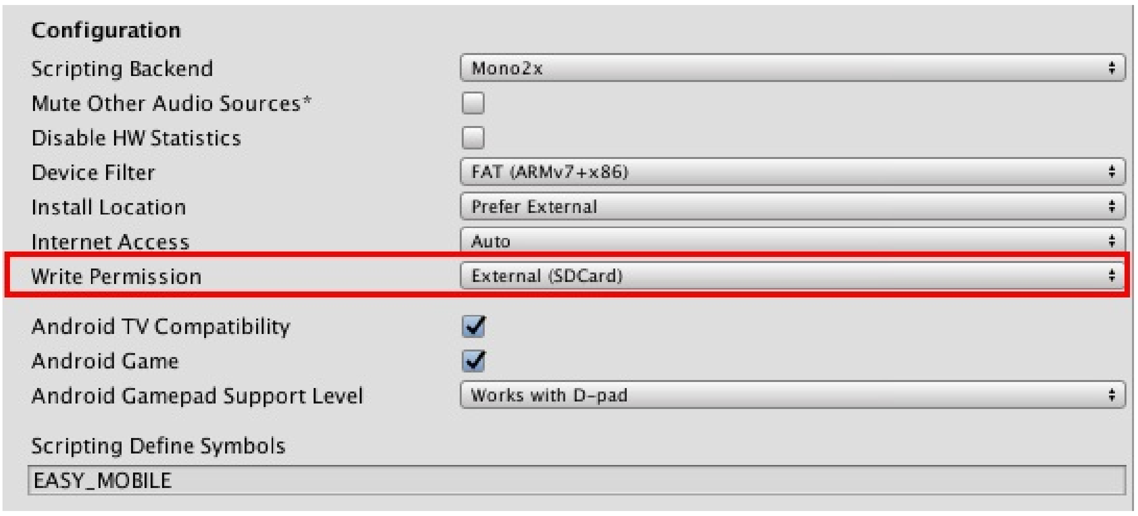
Below are the sharing interfaces on iOS and Android, respectively.





Enable External Write Permission on Android

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



Configuration

Scripting Backend	Mono2x
Mute Other Audio Sources*	☐
Disable HW Statistics	☐
Device Filter	FAT (ARMv7+x86)
Install Location	Prefer External
Internet Access	Auto
Write Permission	External (SDCard)
Android TV Compatibility	✓
Android Game	✓
Android Gamepad Support Level	Works with D-pad

Scripting Define Symbols

EASY_MOBILE

Scripting

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

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

Capture Screenshots

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

Capture and Save a Screenshot as PNG Image

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

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

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

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

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

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

Capture a Screenshot into a Texture2D

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

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

    // Create a Texture2D object of the screenshot using the CaptureScreenshot() metho
d
    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 objec
t of it
IEnumerator CaptureScreenshot()
{
    // Wait until the end of frame
    yield return new WaitForEndOfFrame();

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

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

Sharing

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

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

Share a Saved Image

You can share a saved image by specifying its path.

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

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

Share a Texture2D

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

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

Release Notes

Version 1.0.0

First release.