# OpenIAB Plugin. Open In-App Billing for Android, iOS and Windows Phone 8

*Extensible plugin for OpenIAB (In-App Purchase Billing) integration in Unity3D*

## Table of Contents

## Introduction

Supporting in-app purchases for different platforms and stores is not a simple process. The developer has to study new API for each new store that he/she wants to use in an application.

OpenIAB plugin enables Unity developers to reduce integration and maintenance time. The plugin uses one common interface for 3 mobile platforms: Android, iOS and Windows Phone 8. It's based on OpenIAB library developed by One Platform Foundation team. OpenIAB plugin comes with full source code under Apache 2.0 license.

**Supported Stores:**

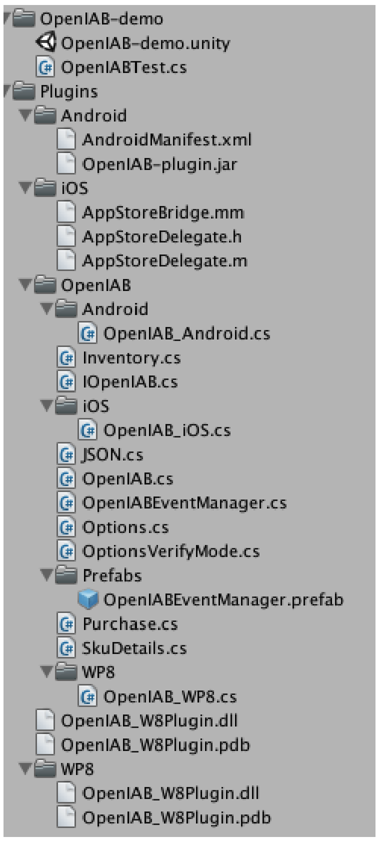
**Android:**

- Google Play
- Amazon Appstore
- Yandex.Store
- Samsung Apps
- Nokia Store
- SlideMe
- Appland
- Apptoide
- AppMall

**iOS:** Apple Store

**Windows phone 8:** Windows Phone Store

## Integration OpenIAB with your Unity project



```
OpenIAB-demo
  OpenIAB-demo.unity
  OpenIABTest.cs
Plugins
  Android
    AndroidManifest.xml
    OpenIAB-plugin.jar
  iOS
    AppStoreBridge.mm
    AppStoreDelegate.h
    AppStoreDelegate.m
  OpenIAB
    Android
      OpenIAB_Android.cs
    Inventory.cs
    IOpenIAB.cs
    iOS
      OpenIAB_iOS.cs
    JSON.cs
    OpenIAB.cs
    OpenIABEventManager.cs
    Options.cs
    OptionsVerifyMode.cs
    Prefabs
      OpenIABEventManager.prefab
    Purchase.cs
    SkuDetails.cs
    WP8
      OpenIAB_WP8.cs
  OpenIAB_W8Plugin.dll
  OpenIAB_W8Plugin.pdb
  WP8
    OpenIAB_W8Plugin.dll
    OpenIAB_W8Plugin.pdb
```

1. Import OpenIAB plugin package into your project.
2. Make sure that Assets->Plugins contains folder structure and files as on screenshot
3. Before using the plugin you need to set up your game in all the stores that you want to support.
4. For iOS: StoreKit library must be added to the XCode project, generated by Unity
5. Now, OpenIAB is ready to use on your project.

## Using OpenIAB

1. Place OpenIABEventManager prefab on your scene.

2. Subscribe to the plugin events in the OnEnable callback. More information about events is described in OpenIAB events section.

```
OpenIABEventManager.billingSupportedEvent += OnBillingSupportedEvent;
OpenIABEventManager.billingNotSupportedEvent += OnBillingNotSupportedEvent;
OpenIABEventManager.queryInventorySucceededEvent+= OnQueryInventorySucceededEvent;
OpenIABEventManager.queryInventoryFailedEvent += OnQueryInventoryFailedEvent;
OpenIABEventManager.purchaseSucceededEvent += OnPurchaseSucceededEvent;
OpenIABEventManager.purchaseFailedEvent += OnPurchaseFailedEvent;
OpenIABEventManager.consumePurchaseSucceededEvent += OnConsumeSucceededEvent;
OpenIABEventManager.consumePurchaseFailedEvent += OnConsumeFailedEvent;
```

3. Map the SKUs for different stores in Start callback.
This step is required when you support stores that have different names for the same SKUs and/or iOS App Store is supported.
E.g. for some reasons your name ***sku_gas*** for Amazon Appstore a***mazon_sku_gas*** and ***google_sku_gas*** for Google Play, it this case mapping is required.
If you use the same names for the all stores that you are going to support and don't use iOS App Store, you can skip this step.

```
private void Start() {
// SKU's for iOS MUST be mapped. Mappings for other stores are optional
OpenIAB.mapSku(SKU_REPAIR_KIT, OpenIAB_iOS.STORE, "30_real");
OpenIAB.mapSku(SKU_GOD_MODE, OpenIAB_iOS.STORE, "noncons_2");
OpenIAB.mapSku(SKU_PREMIUM_SKIN, OpenIAB_iOS.STORE, "noncons_1");

// Map SKUs for Google Play
OpenIAB.mapSku(SKU_REPAIR_KIT, OpenIAB_Android.STORE_GOOGLE, "sku_repair_kit");
OpenIAB.mapSku(SKU_PREMIUM_SKIN, OpenIAB_Android.STORE_GOOGLE, "sku_premium_skin");
OpenIAB.mapSku(SKU_GOD_MODE, OpenIAB_Android.STORE_GOOGLE, "sku_god_mode");

// Map SKUs for Amazon
OpenIAB.mapSku(SKU_REPAIR_KIT, OpenIAB_Android.STORE_AMAZON,
"amazon.sku_repair_kit");
OpenIAB.mapSku(SKU_PREMIUM_SKIN, OpenIAB_Android.STORE_AMAZON,
"amazon.sku_premium_skin");
OpenIAB.mapSku(SKU_GOD_MODE, OpenIAB_Android.STORE_AMAZON, "amazon.sku_god_mode");

// Map SKUs for SlideME
OpenIAB.mapSku(SKU_REPAIR_KIT, SLIDE_ME, "sm.sku_repair_kit");
OpenIAB.mapSku(SKU_PREMIUM_SKIN, SLIDE_ME, "sm.sku_premium_skin");
OpenIAB.mapSku(SKU_GOD_MODE, SLIDE_ME, "sm.sku_god_mode");

// Map SKUs for Yandex.Store
OpenIAB.mapSku(SKU_REPAIR_KIT, OpenIAB_Android.STORE_YANDEX,
"yandex.sku_repair_kit");
OpenIAB.mapSku(SKU_PREMIUM_SKIN, OpenIAB_Android.STORE_YANDEX,
"yandex.sku_premium_skin");
OpenIAB.mapSku(SKU_GOD_MODE, OpenIAB_Android.STORE_YANDEX, "yandex.sku_god_mode");

// Map SKUs for Windows Phone 8
OpenIAB.mapSku(SKU_REPAIR_KIT, OpenIAB_WP8.STORE, "wp8.sku_repair_kit");
OpenIAB.mapSku(SKU_PREMIUM_SKIN, OpenIAB_WP8.STORE, "wp8.sku_premium_skin");
OpenIAB.mapSku(SKU_GOD_MODE, OpenIAB_WP8.STORE, "wp8.sku_god_mode");
}
```

4. Set up Options object.
Using Options you can set up the following settings:

- *checkInventory*
  Store, where the user already bought non-consumable products, will be chosen. **Works only on Android.**
- *prefferedStoreNames*
  If several stores can provide billing for the application on the same device, a store with name specified in the setting will be chosen. **Works only on Android.**
- *verifyMode*
  You can check purchases by yourself (**VERIFY_SKIP**) or delegate it to OpenIAB (**VERIFY_EVERYTHING**, **VERIFY_ONLY_KNOWN**). In the second case you need to pass public keys for supported stores. **Works only on Android.**
- *public keys*
  If verifyMode is set to **VERIFY_EVERYTHING** or **VERIFY_ONLY_KNOWN**, you need to provide public keys. These keys can be obtained from Developer Consoles for certain stores. **Works only on Android and not applicable for all the Android stores.**

```
var options = new OnePF.Options();

options.checkInventory = false;
options.prefferedStoreNames = new string[] { OpenIAB_Android.STORE_YANDEX };
options.verifyMode = OptionsVerifyMode.VERIFY_ONLY_KNOWN;

// Add Google Play public key
options.storeKeys.Add(OpenIAB_Android.STORE_GOOGLE,
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAm+6Tu90pvu2/pdPCI+xcAEoxExJBDYsstQHGl2
8FPeuGjVv/vzguk19WqLcAOHptt5ahYB4LD8PugkMXmgCoYTw0WhWz70kplkkiwVsy9mRPJPsk2F1z/y1w1
76kV6IwdmGKgliRzPLHp2AUo1g+8XrFVF8V9K6n0uVQqfQ5sCEYdRPO+58b5qNG5kJ7wMYCB8ByY/BCddZD
M9mbBziYQIxj/u1Wn45ptHzZv/hlxjHXaqB+UJB1uJZS4fw1w80XPwH7gHWbsVJS6d9fpv2S/nwOIcHmQtQ
2W7SXJRhFbdHrjtpc/LHGfrB4KEthHl2wolFXepeJUjrkM2t5PN7NIwIDAQAB");

// Add SlideME public key
options.storeKeys.Add(SLIDE_ME,
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5p5XkwapZsXrpHvrML6Oac4OuDwGPBfC8j1GMi
Pka0v1MXGN6rcC37qIOOsEMN9v9csS3mLPGINMHmcDJTTrIuLDbSB0QmB7iC3EzfUBAitHghEgDOba0Jn06
tfcMrXalNQ8lpZJh4W1QgwWKra0CUTEHWKGwOdTS6YLQanvsC6B/16iGGFGymkKjGi0ptouplgvwZHe+4gq
o6SoR5tRK7fkcSS+qSzHYdvAcmhzAYGKaV1Ihjy3dd9n2Jz5XeoNag4MSbKQ0YmHyjmyvyKliKOMDps3V5X
9DJzTSSVOSYDVbrFPtdKzr2mJD7T7mtoTnaXYUQLCWOCQs2Oi7djW+QIDAQAB");

// Add Yandex.Store public key
options.storeKeys.Add(OpenIAB_Android.STORE_YANDEX,
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArvbmWP5xYYSKpdnjvLIRWlFzkGT1xoZIekhWPk
GonE+KRd5EWNejLPvOcayY98+NCkLHKSENQSzH+T8IUIEQOhNFzviIblfy3UYG0cFcIehKOV8IiIxvPRAWH
JdzrohTjCCU1A/Lz1NtUn/yxyezrbC0l2ebAdrZSTONMNxMSKAB1+YVbzXI7u4ls9/FIVnEXOyMnCHADEOg
oklFnXEdajJHN/1o+cRz61ow8Qxr+IwG4SSQD1vlv7qkxwEVqBXdhkJBPprZRLq1+VWu+E3f+nYY0I/mHqs
n+pY6z/XYK21gpo4ZbCQzsEDc9ySMeW9mYXZgcIdCTuUic9S26tYfGQIDAQAB");
```

5. Init OpenIAB using the Options objects.

```
OpenIAB.init(options);
```

6. Use provided API for query inventory, to make purchases, etc.

After successful initialization the best practice is to query inventory, to give the user his purchases, made under his account logged into the device (iOS, Windows Phone 8) or the selected store (Android).

```
OpenIAB.queryInventory()
```

If the inventory contains something that should be consumed, call

```
OpenIAB.consumeProduct("sku_name");
```

To start purchase flow, call `OpenIAB.purchaseProduct("sku_name_to_consume");`

```
OpenIAB.purchaseProduct("onepf.sku");
```

Purchase the product with defined SKU.
When the store executes the request, the purchase status will be notified by event.
There are 2 types of events can be:

- purchaseSucceededEvent
- purchaseFailedEvent

You have to show relevant information to the end when when purchaseSucceededEvent is called.

**All these methods are asynchronous, the result will be sent to callbacks, describing in step 2.**

```
OpenIAB.restoreTransactions();
```

**For iOS only.** Restoring previously purchased non-consumable products.
TransactionRestoredEvent is called for every restored transaction. If the result of the process is successful, then restoreSucceededEvent is called, otherwise  restoreFailedEvent is called.

6. When you finish your work with in app purchases, plz disconnect from subscribed application services.

```
OpenIAB.unbindService();
```

| | |
|---|---|
| iOS platform: | While mappings for other stores are optional, SKU's for iOS must be mapped.<br>StoreKit library should be added to the XCode project, which is generated by Unity. |

## Example: Handling Purchases

```
private const string SKU_MEDKIT="sku_medkit";
private const string SKU_AMMO="sku_ammo";
private const string SKU_INFINITE_AMMO="sku_infinite_ammo";
private const string SKU_COWBOY_HAT="sku_cowboy_hat";

//Some game logic refs
[SerializeField] private AmmoBox_playerAmmoBox=null;
[SerializeField] private MedKitPack_playerMedKitPack=null;
[SerializeField] private PlayerHat_playerHat=null;

private void OnPurchaseSucceded(Purchase purchase) {
 //Optional verification of the payload.Can be moved to a custom server
 if(!VerifyDeveloperPayload(purchase.DeveloperPayload))  return;

 //Check purchased SKU
 switch(purchase.Sku) {
  case SKU_MEDKIT: //Consume consumable product
   OpenIAB.consumeProduct(purchase);
   return;
  case SKU_AMMO:
   OpenIAB.consumeProduct(purchase);
   return;
  case SKU_COWBOY_HAT:
    _playerHat.PutOn = true;
    break;
  case SKU_INFINITE_AMMO:
   _playerAmmoBox.IsInfinite = true;
   break;
  default:
  Debug.LogWarning("UnknownSKU:" + purchase.Sku);
  break;
 }
}
```

## Testing In-App Purchases

Here are some useful links that allow to reduce time when verifying in-app purchases feature in your game with different stores.

**IOS:**

https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnectInAppPurchase_Guide/Chapters/TestingInAppPurchases.html

**Windows Phone 8:**

**http://msdn.microsoft.com/en-us/library/windows/apps/jj681689(v=vs.105).aspx**

**Android:**

**Common tips for Android stores:**

1. Android:versionCode attribute value in the AndroidManifest.xml of the local test application must match with the value of your apk in the Developer Console

2. The apk to test must be sign with the same keystore as was used for the apk uploaded to the Developer Console

3. Don't forget to add required permissions and components for the stores to the AndroidManifest.xml

```
<!--Google Play-->
<uses-permission android:name="com.android.vending.BILLING" />
<!--Open Stores-->
<uses-permission android:name="org.onepf.openiab.permission.BILLING" />
<!--Amazon Appstore-->
<receiver android:name="com.amazon.inapp.purchasing.ResponseReceiver">
<intent-filter>
<action android:name="com.amazon.inapp.purchasing.NOTIFY"
android:permission="com.amazon.inapp.purchasing.Permission.NOTIFY"/>
</intent-filter>
</receiver>
<!--Samsung Apps-->
<uses-permission android:name="com.sec.android.iap.permission.BILLING" />
<!--Nokia-->
<uses-permission android:name="com.nokia.payment.BILLING"/>
<!--SlideME-->
<uses-permission android:name="com.slideme.sam.manager.inapp.permission.BILLING" />
```

4. If you use Proguard for your application, don't forget to add

| | |
|---|---|
| Google Play | -keep class com.android.vending.billing.* |
| Amazon AppStore | -dontwarn com.amazon.** -keep class com.amazon.** {*;} -keepattributes *Annotation* -dontoptimize |
| Samsung Apps | -keep class com.sec.android.iap.** |
| Nokia Store | -keep class com.nokia.payment.iap.aidl.** |

5. The app installer package can be set using the following instruction:

```
adb install -i installer.package.name /path/to/YourApp.apk
```

**Google Play:**

http://developer.android.com/google/play/billing/billing_testing.html

| | |
|---|---|
| Note from Google: | *"We recently made some changes to our systems and we are now requiring an app to be published before testing.*<br><br>*We are currently recommending to publish your APK to the Alpha channel in order to test licensing, in-app billing, and expansion files. There is no need to create a special testing group in the Alpha channel to test these features, however the app must be published and not in draft mode.*<br><br>*We apologize for the inconvenience and are working to update our documentation to reflect these changes."* |

**Amazon Appstore:**

https://developer.amazon.com/appsandservices/apis/earn/in-app-purchasing/docs/testing-iap

**Slide.me:**

http://slideme.org/developers/iap

## OpenIAB methods

In-app purchase functionality is handled by OpenIAB class.

```
// mapSku does association store product name with SKU
//@param sku product ID
//@param storeName name of the store
//@param storeSku product ID in the store
//this method must be only called before init
public static void mapSku(string sku, string storeName, string storeSku)

//Starts up the billing service. This will also check to see if in app billing is supported
//and fire the appropriate event
//@param options library options instance like Dictionary<string, string>
public static void init(Options options)

//Unbinds and shuts down the billing service
public static void unbindService()

//Checks if subscriptions are supported. Currently used on Android  only
//@return true if subscriptions are supported on the device
public static bool areSubscriptionsSupported()

//Sends a request to get all completed purchases
public static void queryInventory()

//Sends a request to get all completed purchases and specified SKUs information
//@param skus product IDs
public static void queryInventory(string[] skus)

//Purchases the product with the given SKU and developerPayload
//@param sku product ID
//@param developerPayload payload to verify transaction
public static void purchaseProduct(string sku, string developerPayload = "")

// Purchases the subscription with the given SKU and developerPayload
//@param sku product ID
//@param developerPayload payload to verify transaction
public static void purchaseSubscription(string sku, string developerPayload = "")

//Sends out a request to consume the product
//"You must send a consumption request for the "owned" in-app product before store
//makes it available for purchase again. Consuming the in-app product reverts it to the
//"unowned" state, and discards the previous purchase data.
//@param purchase purchase data holder
public static void consumeProduct(Purchase purchase)

//Restore purchased items.
//iOS AppStore requirement.
public static void restoreTransactions()
```

```
//Is verbose logging enabled
//@return true if logging is enabled
public static bool isDebugLog()

//Get more debug information
//@param enabled if logging is enabled
public static void enableDebugLogging(bool enabled)

//Get more debug information
//@param enabled if logging is enabled
//@param tag Android log tag
public static void enableDebugLogging(bool enabled, string tag)
```

## OpenIAB events

```
//Successful Init callback.
//Billing is supported on current platform
public static event Action billingSupportedEvent;

//Failed Init callback. Billing is not supported on current platform
public static event Action<string> billingNotSupportedEvent;

//Successful QueryInventory callback. Purchase history and store listings are returned
public static event Action<Inventory> queryInventorySucceededEvent;

//Failed QueryInventory callback.
Public static event Action<string> queryInventoryFailedEvent;

//Successful purchase callback. Fired after purchase of a product or a subscription
public static event Action<Purchase> purchaseSucceededEvent;

//Failed purchase callback
public static event Action<int, string> purchaseFailedEvent;

//Successful consume attempt callback
public static event Action<Purchase> consumePurchaseSucceededEvent;

//Failed consume attempt callback
public static event Action<string> consumePurchaseFailedEvent;

//Fired when transaction was restored
public static event Action<string> transactionRestoredEvent;

//Fired when transaction restoration process failed
```

```
public static event Action<string> restoreFailedEvent;

//Fired when transaction restoration process succeeded
public static event Action restoreSucceededEvent;
```

## Suggestions/Questions

We seek to constantly improve our project and give Unity developers more power when working with in-app purchases. We are open to any comments and suggestions you may have regarding the additional features to be implemented in our plugin.

If you know about issues we missed, please let us know in
Issues on GitHub: https://github.com/onepf/OpenIAB/issues
or by email: unitysupport@onepf.org

If you detect some issues with integration into your project, please let us know in
http://stackoverflow.com/questions/ask/advice?

| | |
|---|---|
| Note: | When you post a question on stackoverlow, mark your post with the following tags "**in-app purchase**", "**unity3d**", "**openiab**". It will help you to get a faster response from our community. |

## License

The source code of the OpenIAB Unity plugin and OpenIAB library are available under the terms of the Apache License, Version 2.0:
http://www.apache.org/licenses/LICENSE-2.0

The OpenIAB API specification and the related texts are available under the terms of the Creative Commons Attribution 2.5 license:
http://creativecommons.org/licenses/by/2.5/