



OPEN IAB UNITY PLUGIN INTEGRATION GUIDE FOR SAMSUNG GALAXY APPS

Version 01

Though every care has been taken to ensure the accuracy of this document, Samsung Electronics Co, Ltd. cannot accept responsibility for any errors or omissions or for any loss occurred to any person, whether legal or natural, from acting, or refraining from action, as a result of the information contained herein. Information in this document is subject to change at any time without obligation to notify any person of such changes.

Samsung Electronics Co, Ltd. may have patents or patent pending applications, trademarks copyrights or other intellectual property rights covering subject matter in this document. The furnishing of this document does not give the recipient or reader any license to these patents, trademarks copyrights or other intellectual property rights.

No part of this document may be communicated, distributed, reproduced or transmitted in any form or by any means, electronic or mechanical or otherwise, for any purpose, without the prior written permission of Samsung Electronics Co, Ltd.

The document is subject to revision without further notice.

All brand names and product names mentioned in this document are trademarks or registered trademarks of their respective owners.

Copyright© 2015 Samsung Electronics, Co., Ltd

All rights reserved.

Purpose and Audience

This document provides the overview, environment setup, prerequisites, plugin integration steps, and common errors for using the OpenIAB Unity Plugin for Samsung IAP 3.0.

This programming guide is designed for developers with knowledge on Unity app development. For details, see the Unity website.

Revision History

The following are highlights of the changes to this document.

Version	Primary Authors	Change Highlights
01	Samsung Research Institute Philippines Media Solution Center America, Samsung Electronics America	Initial Draft

Contents

1. Overview	6
1.1. Unity App and Samsung IAP Architecture.....	7
1.2. Development, Distribution, and Management	7
To distribute and manage your service app and items via Samsung systems:.....	8
2. Development Environment	9
2.1.1. Unity	9
2.1.2. MonoDevelop.....	9
3. OpenIAB Plugin Integration.....	10
3.1. Building the OpenIAB Unity Plugin.....	10
To build the OpenIAB plugin from source code:.....	10
3.2. Importing the OpenIAB Unity Plugin	12
To import the OpenIAB Unity plugin into your Unity project:	12
3.3. Configuring and Initializing the OpenIAB Unity Plugin.....	14
To configure and initialize the OpenIAB plugin for your service app environment:	15
3.4. Integrating OpenIAB Item List and Purchase APIs.....	20
3.4.1. Commercial Item Types	20
3.4.2. Available Item and User-Purchased Item Lists	21
To get a list of all items available in the service app:.....	22
To generate and get a list of user-purchased items:.....	24
To get the locally cached list of user-purchased items:	27
3.4.3. Purchasing Commercial Items	28
To purchase a consumable item:	28
To purchase a non-consumable item:	30
To purchase an auto-recurring subscription item:.....	31
To purchase a non-recurring subscription item:	32
3.5. Terminating Commercial Item Processing	34
3.5.1. Unsubscribe to Plugin Events	34
3.5.2. Unbind from the IAP Billing Service	34
3.6. Building Your Service App	35
To build your Unity service app:.....	35

3.7. Testing Your Service App Integration	36
3.7.1. Service App Operating Mode	36
3.7.2. OpenIAB Plugin Error Returns and Integration Errors	36
A. Integration Interdependencies	38
A.1. Item Purchase Processing Modes	38
A.2. Server-to-Server Purchase Verification.....	39
To perform server-to-server purchase verification:.....	39
B. Service App and Item Registration, Certification, and Management	41
To initially register your development service app and items with Seller Office:.....	41
To configure your service app for the commercial production environment:	41
To register your production service app and items with Seller Office:.....	42
To certify your production service app and items:.....	42
To manage your distributed service app and items:.....	42
C. App Store Selection for Item Purchase	43
To determine the app store that an item is to be purchased from:	43

Figures

Figure 1	OpenIAB Unity Plugin Block Diagram	7
Figure 2	API Call Sequence: OpenIAB Plugin Configuration and Initiation.....	14
Figure 3	API Call Sequence: Generating a List of All Available Commercial Items	21
Figure 4	API Call Sequence: Generating and Getting a List of User-Purchased Items....	24
Figure 5	API Call Sequence: Getting the Cached List of User-Purchased Items	27
Figure 6	API Call Sequence: Purchasing Commercial Items	28
Figure 7	Server-to-Server Verification	39

1. Overview

The modified Open In-App Billing (OpenIAB) Unity plugin and library extend OpenIAB purchasing and billing transaction management by relying on the Samsung In-App Purchase (IAP) 3.0, including its latest features:

- **Auto-recurring subscriptions support**
Developers can sell in-app items every month with automatically recurring billing.
- **Easier service app identification**
Identification is based on the service app package name instead of a 12-digit item group ID.
- **Custom commercial item ID support**
Commercial item IDs can be conveniently managed and identified using a custom item name instead of a 12-digit product item ID.
- **Faster lists of user-purchased items**
Client-side caching allows faster retrieval of a list of user-purchased items for the specified item ID.

The OpenIAB plugin enables Unity developers to reduce turnaround time to integrate in-app billing and purchasing functionality. The OpenIAB library is a common interface for mobile platforms (such as the Android OS, iOS, and Windows Phone OS).

By integrating OpenIAB API calls into service app logic and code, developers can distribute their service app via different OpenIAB-compliant app stores (such as Samsung Galaxy App Store and Google Play Store) and their associated systems for commercial purchasing, billing, and management.

NOTE: This document describes integration and the use of the Samsung app store and systems.

When your service app integrates OpenIAB API calls AND selects Samsung Galaxy App Store as the current app store, the following components are used:

Galaxy Apps Store	Service app distribution (either free or commercial distribution)
Samsung IAP	Commercial item purchasing and listing functionality
Samsung Account	User validation and authorization for downloading and purchasing service apps and commercial items Samsung Account, invoked by Samsung IAP, does not have to be integrated by OpenIAB integration developers.
Samsung Galaxy Apps Seller Office	UI management (including service app and commercial item registration, price management, distribution and sales management, financial accounting)
Samsung Billing	Fundamental support for purchasing, billing, and financial accounting

Samsung IAP functionality, via the OpenIAB API, makes it possible for developers to sell virtual items, services, and subscriptions (*commercial items* or *items*) from within the Android service apps that are freely or commercially distributed via the Samsung Galaxy Apps Store.

1.1. Unity App and Samsung IAP Architecture

The OpenIAB Unity Plugin package is integrated into your Android service app together with the Unity Framework to use the modified OpenIAB library to invoke Samsung IAP 3.0 functionality.

As shown in Figure 1, the OpenIAB library is wrapped by the Unity plugin to be able to use it in Unity. The Android service app uses OpenIAB to communicate with the Samsung IAP service package via the Android Interface Definition Language (AIDL).

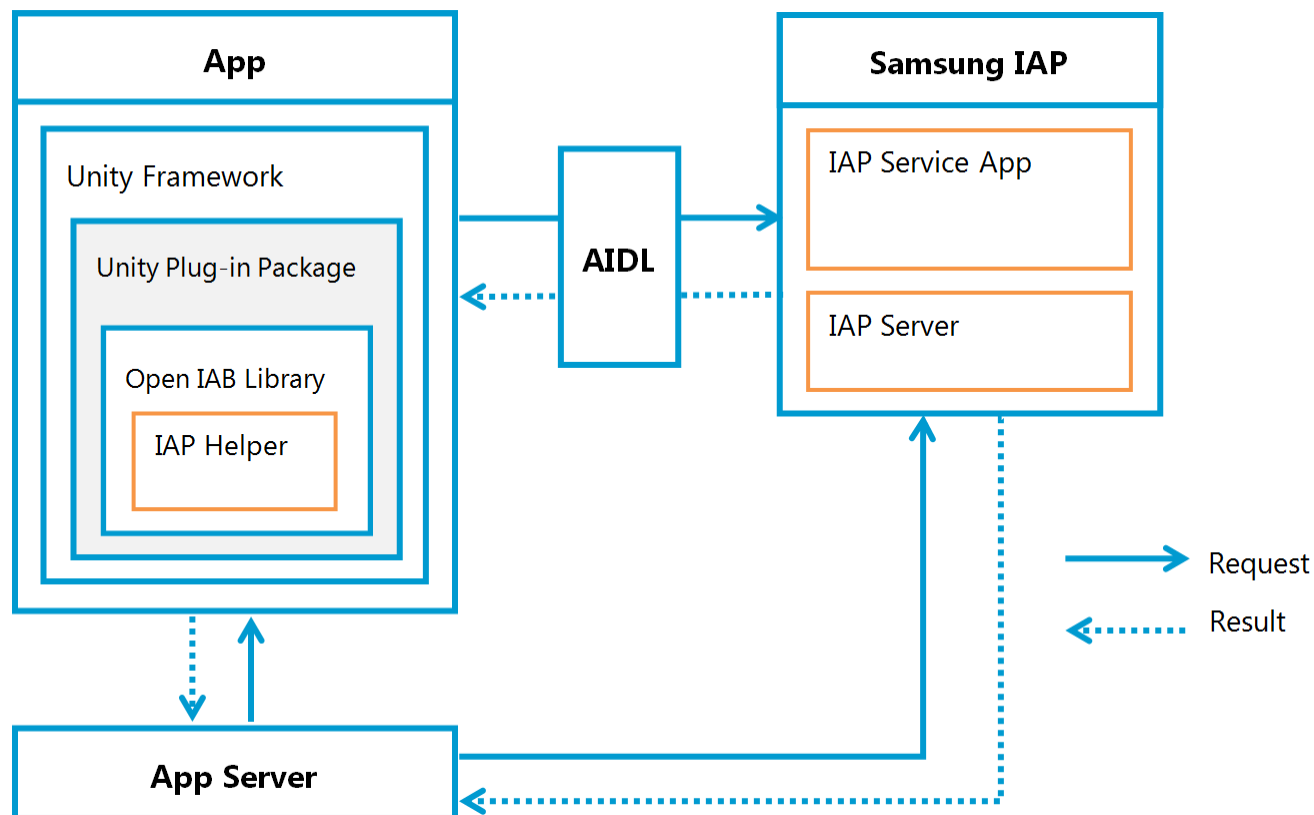


Figure 1 OpenIAB Unity Plugin Block Diagram

1.2. Development, Distribution, and Management

This section presents the high-level overview of how to integrate the OpenIAB plugin to support commercial item processing; register, certify, and distribute your service app and its commercial items; and manage purchasing, billing, and sales of your service app and its commercial items via the Samsung Galaxy Apps Store and Samsung systems.

For an overview of commercial item registration and management, see [How to Start Samsung IAP SDK](#).

To distribute and manage your service app and items via Samsung systems:

1. Develop your service app and its commercial items:

- Ensure your service app and its items meets Samsung certification requirements and guidelines. For details, see the *Samsung Galaxy Apps Seller Office Service Application Certification Guide*.
- Set up your development environment:
 - Use the [Unity](#) game engine to develop your service app.
 - Install [MonoDevelop](#) in your development environment and use it to write service app scripts.
 - Base your service app on [JDK 1.6](#) or higher.
- Initially register your development service app and its commercial items with the Samsung Galaxy Apps Seller Office. (In this step, you will NOT submit your service app for certification. This step enables development integration testing. In step 2, you will completely register your production version and submit it for certification to be distributed.)
 - Your service app representative(s) must get a Samsung Account user account.
 - Your service app representative(s) must get a Seller Office commercial seller account. For details, see the *Samsung Galaxy Apps Seller Office User Account Guide*.
 - Initially register your service app and its commercial items (even though they are in development). For details, see [Appendix B](#).
- Integrate OpenIAB plugin API calls into your service.
 - Your service app must NOT be a Gear application or a KNOX 2.0 application.
 - Your service app must properly integrate Open IAB API calls to support the commercial item listing and purchasing functionality of Samsung IAP.
 - Test your integrated service app.
 - For details, see section 3. OpenIAB Plugin Integration in this document.

2. Distribute your service app and its commercial items via the Samsung Galaxy Apps Store:

- Change the development version of your service app and items into their normal, production version, including adding production version commercial items. For details, see [Appendix B](#).
- Update the Seller Office registration with your production service app and items, which includes but is not limited to the following:
 - Ensure all service app and commercial item prices are correct.
 - Ensure the countries your service app is to be distributed to are specified.
 - Ensure the user devices that your service app is to be distributed to are specified. Distribution devices must meet the following requirements:
 - The device must be a Samsung device with the following clients installed: Samsung Account Client, Samsung Galaxy Apps Store Client, Samsung Billing Client, and Samsung IAP.
 - The device must be selected for distribution in the Samsung Galaxy Apps Seller Office.
 - For details, see [Appendix B](#).

- Submit your service app and its items for certification via the Samsung Galaxy Apps Seller Office. For details, see [Appendix B](#).
3. Manage service app and item distribution, sales, and finances via the Samsung Galaxy Apps Seller Office:
- Generate and review distribution and sales reports.
 - Generate and review financial accounting reports.
 - For details, see [Appendix B](#).

2. Development Environment

To integrate the OpenIAB plugin during service app development, the following are needed.

2.1.1. Unity

Download the Unity game engine version 5.2 from the Unity website. After creating your service app, you need to set the platform to Android and build the application to export as an APK, so that its binaries can be registered with the Samsung Galaxy Apps Seller Office.

2.1.2. MonoDevelop

To write scripts for your service app, download and install MonoDevelop into your service app development environment. C# code scripts are used to integrate OpenIAB plugin API calls. For MonoDevelop installation details, see the MonoDevelop website.

3. OpenIAB Plugin Integration

This section describes how to successfully integrate the OpenIAB Unity plugin into your Unity project, using the sample game service app *City Flyer*, a three-dimensional plane flight simulator that lets users navigate over a city. City Flyer makes OpenIAB API calls to invoke on Samsung IAP commercial item functions (purchasing items and getting lists of items), which rely on Samsung systems (such as Samsung Billing and Seller Office) to support IAP and provide the service app team with management and control.

NOTE: There may be potential conflicts when the OpenIAB Unity plugin is used in conjunction with other billing plugins.

3.1. Building the OpenIAB Unity Plugin

The OpenIAB plugin can be downloaded from:



<https://github.com/Samsung-US-Developer-Support/Samsung-OpenIAB-Unity-Plugin>.

If you are able to download the built plugin successfully, skip the procedure in this section.

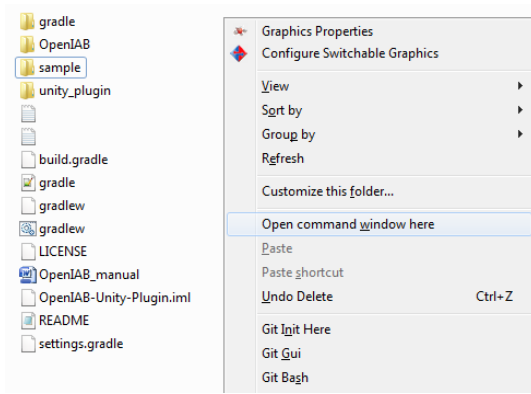
In case you experience problems in downloading the built OpenIAB plugin, you can download the library source code and build the plugin manually. If there is an updated version of the plugin package and plugins source code, you have to download the latest plugin package or build it from the updated source code.

To build the OpenIAB plugin from source code:

1. Download the OpenIAB library from <https://github.com/Samsung-US-Developer-Support/Samsung-OpenIAB-Unity-Plugin>
2. Paste the library in the root level of the project folder.

Name	Date modified	Type	Size
 OpenIAB Unity Plugin	2015-11-05 오후 5:...	File folder	
 Unity Sample App	2015-11-05 오후 5:...	File folder	

3. In the OpenIAB Unity Plugin folder, press **Shift** then right-click and select **Open Command Window here**



4. Enter the following command:

```
gradlew clean buildPlugin
```

5. After a successful build, the Unity package file can be found in the `unity_plugin/build/outputs` folder.

```

:Unity Plugin:dexRelease
:Unity Plugin:processReleaseJavaRes UP-TO-DATE
:Unity Plugin:packageRelease
:Unity Plugin:assembleRelease
:Unity Plugin:buildPlugin
Unity executable: C:\Program Files\Unity\Editor\Unity.exe
Unity package source root: D:\Workspaces\Special Project Workspace\Unity IAP\OpenIABUnityPlugin\branches\dev\OpenIAB Unity Plugin\unity_plugin\unity_src
Unity package build: D:\Workspaces\Special Project Workspace\Unity IAP\OpenIABUnityPlugin\branches\dev\OpenIAB Unity Plugin\unity_plugin\build\outputs
BUILD SUCCESSFUL
Total time: 1 mins 53.785 secs

```

NOTE: You can modify the build path of the Unity package file by changing the `unityExecutable` property in the `gradle.properties` file.

3.2. Importing the OpenIAB Unity Plugin

Before importing and using the plugin, ensure that you have registered your service app and its commercial items in the Samsung Apps Seller Office, regardless of whether your application is in production, development, or testing phases.

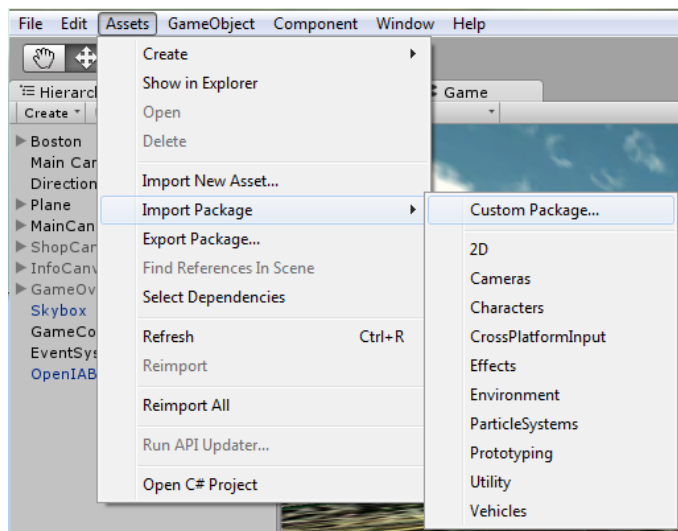
After getting a local copy of a built OpenIAB Unity plugin, you can import and use it.

To import the OpenIAB Unity plugin into your Unity project:

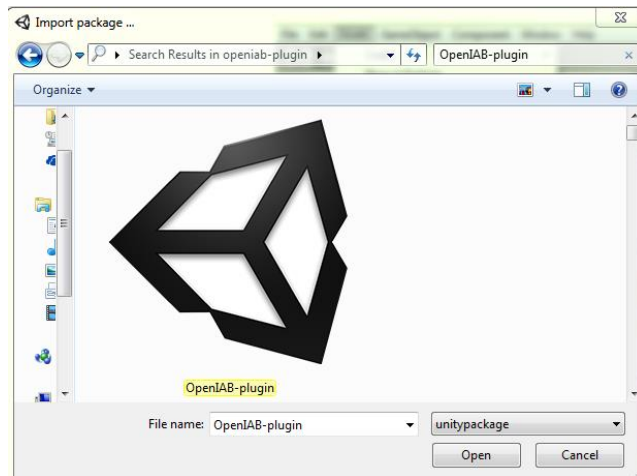
1. Start up the Unity and open the project.
2. Import the plugin by EITHER:
 - a. Drag and drop:
 - i. Open the window with the local copy of the plugin file.
 - ii. Drag and drop the plugin file to your current Unity window.

OR

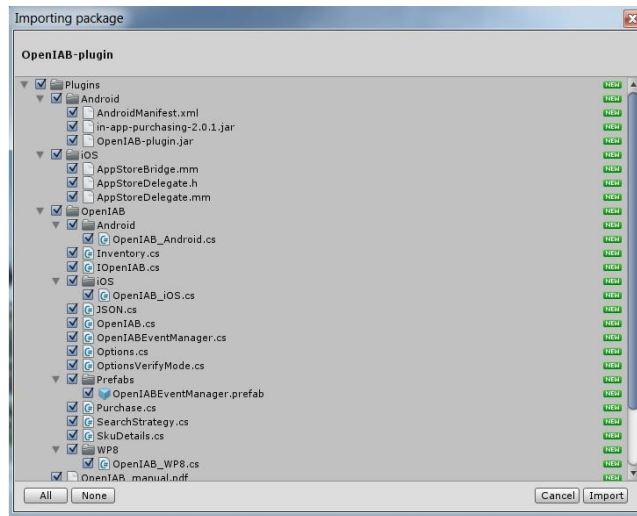
- b. File selection:
 - i. Click **Assets > Import Package > Custom Package**



- ii. Navigate to the Unity package file **OpenIAB-plugin.unpackage** and click **Open**



iii. In the Importing Package Window, check the **Plugins** directory and click **Import**



3.3. Configuring and Initializing the OpenIAB Unity Plugin

This section describes how to set up the OpenIAB plugin functions in the City Flyer source code, and initialize the plugin.

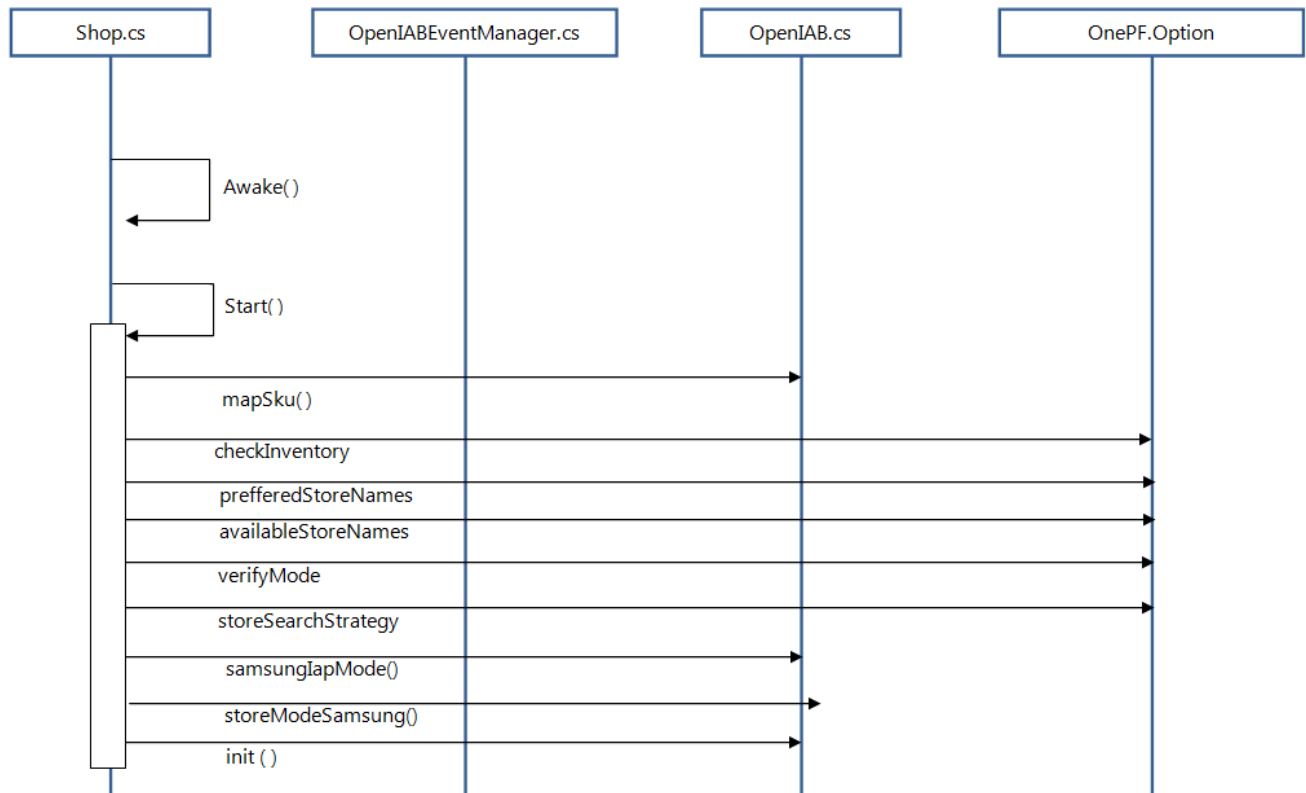
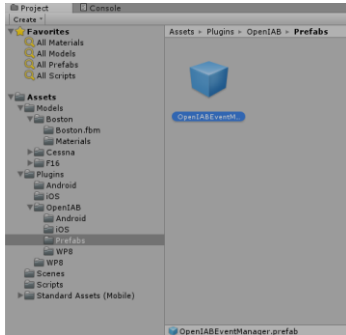


Figure 2 API Call Sequence: OpenIAB Plugin Configuration and Initiation

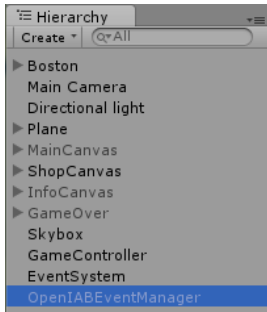
Figure 2 illustrates the sequence of method calls to configure and initialize the OpenIAB plugin. The components are described in item 2 of the procedure below.

To configure and initialize the OpenIAB plugin for your service app environment:

1. Set OpenIAB plugin event manager to allow your service app to subscribe / unsubscribe to plugin events:
 - a. Drag the `OpenIABEventManager.prefab` file found in the `Assets/Plugins/OpenIAB/Prefabs` folder.



- b. Drop the file in to your scene.



2. Subscribe to the plugin purchase and billing events using the OpenIABEventManager by registering listeners.

The `OnEnable` callback function, implemented in the `Awake()` function of `Shop.cs.`, manages the plugin events. In Unity, the service app calls `Awake()` when the script instance is being loaded.

For more class and method details, see the *OpenIAB Unity Plugin for Samsung IAP 3.0 API Reference*.

Shop.cs

```
private void Awake()
{
    // Subscribe to all billing events
    OpenIABEventManager.billingSupportedEvent += OnBillingSupported;
    OpenIABEventManager.billingNotSupportedEvent += OnBillingNotSupported;
    OpenIABEventManager.queryInventorySucceededEvent +=
        OnQueryInventorySucceeded;
    OpenIABEventManager.queryInventoryFailedEvent += OnQueryInventoryFailed;
    OpenIABEventManager.purchaseSucceededEvent += OnPurchaseSucceeded;
    OpenIABEventManager.purchaseFailedEvent += OnPurchaseFailed;
    OpenIABEventManager.consumePurchaseSucceededEvent +=
        OnConsumePurchaseSucceeded;
    OpenIABEventManager.consumePurchaseFailedEvent +=
        OnConsumePurchaseFailed;
    OpenIABEventManager.transactionRestoredEvent += OnTransactionRestored;
    OpenIABEventManager.restoreSucceededEvent += OnRestoreSucceeded;
    OpenIABEventManager.restoreFailedEvent += OnRestoreFailed;
}
```

3. Map service app item SKUs:

OpenIAB allows you to write a single block of code that would work with any popular app store. However, some app stores might have different naming conventions for their items. The SKUs (Store Keeping Units) are developer-defined strings that are used by OpenIAB for standardization. The SKUs for all commercial items available in your service app must be mapped to each supported app store in the OpenIAB Start callback method. In Unity, Start() is invoked before the first frame update. It calls OpenIAB's mapSku() functions, which maps the item SKUs to the specified app stores.

For Samsung IAP, commercial item SKUs are the item IDs, which are specified by your service app team and registered with the Samsung Galaxy Apps Seller Office:

Item ID	Item Title	Item Type	Duration	Prices (\$)
sku_ghost_mod...	Ghost Mode	Non-recurring subscriptions	1 Month	10
sku_infinite_...	Infinite Fuel	Auto-recurring subscriptions		10
sku_change_pl...	New Plane	Non-consumable		5
sku_plane_fue...	sku_plane_fuel	Consumable		2

NOTE: Previously, Samsung Galaxy Apps Store used a 12-digit item group ID to identify all items within the app. This is deprecated in IAP 3.0. Now, the service app package name automatically identifies the service app. Your service app can still use the item group ID to identify the items' service app (for example, if your service app previously used it).

Shop.cs

```
private void Start () {

    /* Map SKU
    * Samsung IAP ver. 3 doesn't require the twelve digit itemGroupID
    * but it can still be used as shown with SKU_PLANE_FUEL */

    OpenIAB.mapSku(SKU_PLANE_FUEL, OpenIAB_Android.STORE_SAMSUNG,
                   "100000105847/sku_plane_fuel");
    OpenIAB.mapSku(SKU_CHANGE_PLANE, OpenIAB_Android.STORE_SAMSUNG,
                   "sku_change_plane");
    OpenIAB.mapSku(SKU_INFINITE_FUEL, OpenIAB_Android.STORE_SAMSUNG,
                   "sku_infinite_fuel");
    OpenIAB.mapSku(SKU_GHOST_MODE, OpenIAB_Android.STORE_SAMSUNG,
                   "sku_ghost_mode");
    ...
}
```

NOTE: The setup above is only for Samsung Galaxy Apps store.

4. Set up the OpenIAB plugin library options:

Library options are specified in `Initialize OnePF.Options()` object. To customize options for initializing OpenIAB, start by creating a `OnePF.Options()` object inside the `Start()` method of `Shop.cs`.

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

The `Options` class is a way to configure how OpenIAB selects and initializes an app store. Since app stores may behave differently or may have unique features, some options are not applicable for all Android app stores. The following library options are only supported by Android-based service apps.

NOTE: Many option settings used to determine the app store to purchase an item from can override and be overridden by other factors. If certain conditions for an app store have been met, then that app store would be chosen instead of other options. For more details about the store option priority and selection process, see Appendix C.

checkInventory Controls the app store that an item is to be purchased from:

- true** Determines the app store that the user previously purchased items from and uses it for the rest of the session.
- false** *Default* Determines and uses the first app store with billing available (not previous user purchases), which makes the app store selection faster but adds the risk of initializing the wrong app store or being unable to find a suitable app store.

```
options.checkInventory = false;
```

storeSearchStrategy Controls the app store that an item is to be purchased from:

- INSTALLER** *Default* Purchase the item either: from the app store that the app was downloaded from, OR from the app store specified when side loading via ADB.

```
var options.storeSearchStrategy =  
SearchStrategy.INSTALLER;
```

Currently, Samsung does not support service app sideloading. The following sideload for Google Play Store can be used.

```
adb install -i com.android.vending <apk name>
```

- BEST_FIT** Purchase the item from the best app store as determined by past purchases and purchasable items.

```
var options.storeSearchStrategy =  
SearchStrategy.BEST_FIT;
```

INSTALLER_THEN_BEST_FIT

Purchase the item either:

- From the app store that the app was downloaded from, OR from the app store specified when side loading via ADB OR when an app store cannot be determined above:
- From the best app store as determined by past purchases and purchasable items

```
var options.storeSearchStrategy =  
SearchStrategy.INSTALLER_THEN_BEST_FIT;
```

preferredStoreNames Controls the app store that an item is to be purchased from:

[App store name(s)]

1 or more app stores to be checked for service availability.

If the item's service app is available from the first listed app store, then the item is purchased from that app store. If the item's service app is NOT available from the first listed app store, then the second listed app store is checked, and so on.

If the item's service app is NOT available from ANY listed app store, then it returns a billing unavailable error.

```
options.preferredStoreNames = new string[] {  
OpenIAB_Android.STORE_SAMSUNG};
```

verifyMode Controls whether and how each item purchase is checked. Verification requires RSA key pairs for supported Android app stores to verify the item.

VERIFY_SKIP Do NOT check item purchases.

NOTE: Samsung IAP does not provide public keys.
Use VERIFY_SKIP when the current app store is Samsung.

VERIFY EVERYTHING OpenIAB checks all item purchases.

VERIFY_ONLY_KNOWN OpenIAB checks only the known item purchases.

```
options.verifyMode = OptionsVerifyMode.VERIFY_SKIP;
```

For testing purposes, the Samsung Galaxy Apps Store can be forced to be selected. This is done through the following method:

```
OpenIAB.storeModeSamsung(boolean enabled);
```

If the parameter is set to true, the Samsung Galaxy Apps Store is selected during OpenIAB initialization.

```
OpenIAB.storeModeSamsung(true); // Forced to Samsung Galaxy Apps Store
```

If the parameter is set to false, the app store is selected depending on the options specified.

```
OpenIAB.storeModeSamsung(false);
```

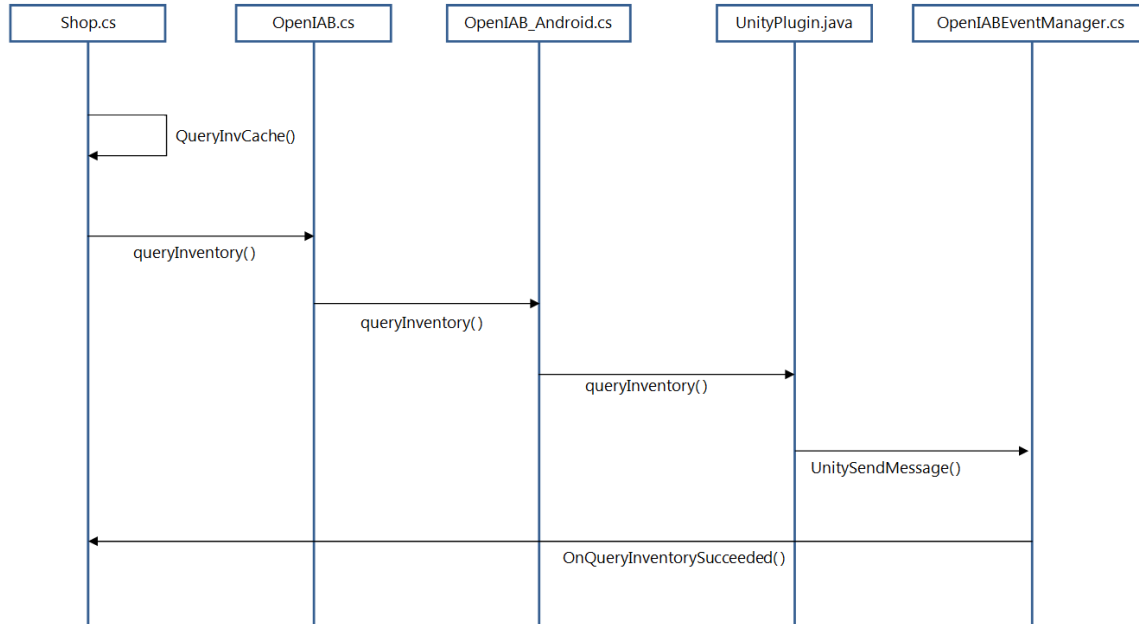
```
Shop.cs
private void Start () {
    ...
    // Set library options
    var options = new OnePF.Options();
    options.checkInventory = false;
    options.storeSearchStrategy = SearchStrategy.BEST_FIT;
    OpenIAB.samsungIapMode(1);
    OpenIAB.samsungForcedMode(true);
    OpenIAB.init(options);
    ...
}
```

5. Initialize the OpenIAB object using the created options object instance, which is implemented in the Start() function.

```
Shop.cs
private void Start () {
    ...
    // Initialize OpenIAB with options defined above
    OpenIAB.init(options);
    ...
}
```

3.4. Integrating OpenIAB Item List and Purchase APIs

The best practice after a successful initialization is to query the item inventory, in order to give the users their purchases made under their logged account in the selected app store. For details about the functions to use, see section 1.1 of the OpenIAB API Reference.

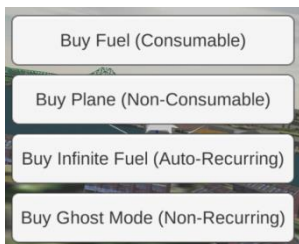


Only purchases that are successfully completed can be verified via Samsung IAP. For added security, it is recommended to have an app server verify a purchase via the verify URL of the Samsung IAP. See section A.3 for more information on Server-to-Server verification.

3.4.1. Commercial Item Types

Samsung IAP 3.0 supports four types of commercial items, which City Flyer game app users can buy:

- **Consumable** Items that can be repurchased multiple times.
- **Non-Consumable** Items that can be used permanently and cannot be repurchased.
Non-consumable items can be recovered even after re-installing a service app.
- **Non-Recurring** Items can be repurchased after a certain period.
- **Auto-Recurring** Items are automatically purchased and billed every specified period.



3.4.2. Available Item and User-Purchased Item Lists

Service apps (for example, in response to user input) can generate and get a list of ALL commercial items currently available to them. These items and their related information (such as price) are currently registered to the service app via the Samsung Galaxy Apps Seller Office.

Service apps (for example, in response to user input) can use either of 2 methods to get a list of items that the current user has purchased up to the time the list was generated:

1. By requesting Samsung systems to generate a new list that is then locally cached and made available to the service app.
2. By retrieving a locally cached list that was generated the last time the list was generated by the first method. This second method is faster than the first method.

3.4.2.1. Getting a List of All Items Available in the Service App

Service apps and their users can get a list of all available items currently registered to the Seller Office account of the service app.

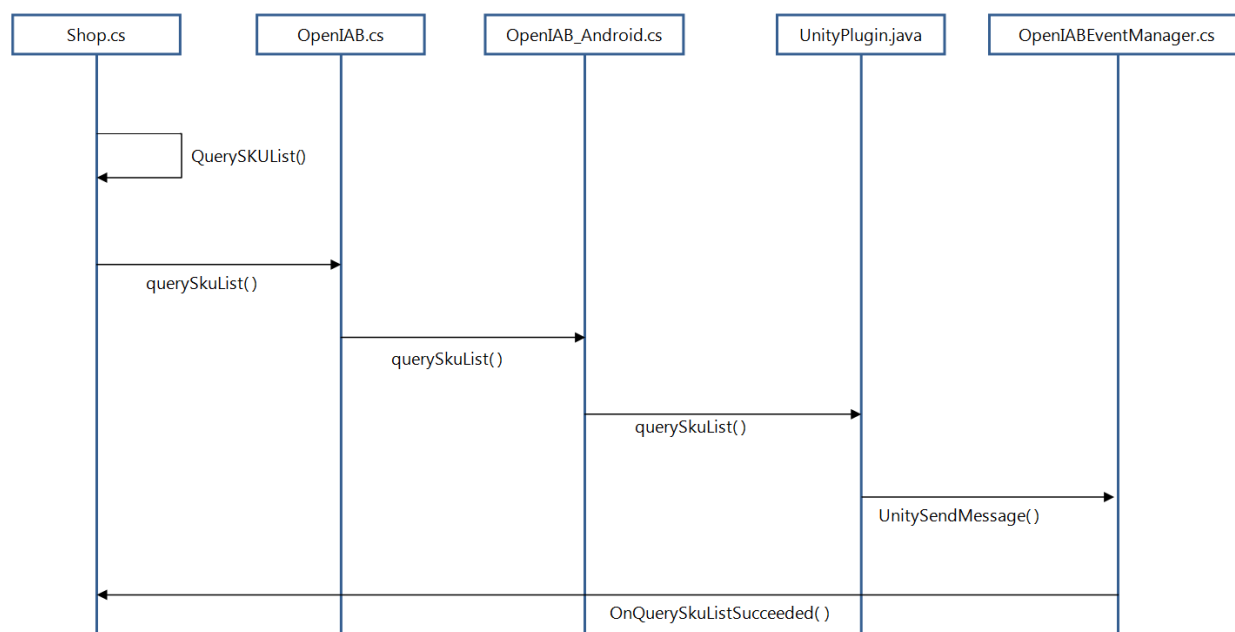
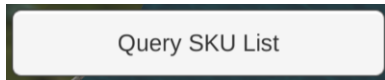


Figure 3 API Call Sequence: Generating a List of All Available Commercial Items

To get a list of all items available in the service app:

1. In the City Flyer UI, click **Query SKU List**



The item names are displayed on the left side of the screen.

2. The button click invokes the function `QuerySKUList()` of `Shop.cs`, which calls `OpenIAB.querySkuList()`.

CAUTION: `querySKUList()` is an `OpenIAB` method which calls a Samsung IAP method, `getItemList()`, and is to be used when the current app store is Samsung. Using this method with other current app stores will throw an `IabException` that should be handled by the service app logic and code. It fires the event `querySkuListFailedEvent()`. The exception can be handled there.

Shop.cs

```
// Called on the onClick() method of the Query SKU List button
// Starts the process to display a list of all available items for purchase
// listed in the Samsung sellers account for this application

public void QuerySKUList(){
    OpenIAB.querySkuList();
}
```

3. `OpenIAB.querySkuList()` calls the `querySkuList()` function of `OpenIAB_Android.cs` for Android devices.

OpenIAB.cs

```
public static void querySkuList() {
    _billing.querySkuList();
}
```

4. `OpenIAB_Android.querySkuList()` uses a Java Native Interface (JNI) to access the `querySkuList()` method in the `UnityPlugin.java` file (which is similar to `OpenIAB_Android.queryInventory()`).

OpenIAB_Android.cs

```
public void querySkuList()
{
    if (!IsDevice())
    {
        return;
    }
    IntPtr methodId = AndroidJNI.GetMethodID(_plugin.GetRawClass(),
        "querySkuList", "()V");
    AndroidJNI.CallVoidMethod(_plugin.GetRawObject(), methodId, new jvalue[] {
    });
}
```

5. After the item list has been retrieved from the IAP server, `onQuerySkuListFinished()` is invoked.
 - a. When the list generation query is successful:
 - i. `onQuerySkuListFinished()` sends the message with the list back to `Shop.cs`.

```
UnityPlugIn.java
public void querySkuList() {
    UnityPlayer.currentActivity.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            _helper.querySkuListAsync(_querySkuListFinishedListener);
        }
    });
}

IabHelper.QuerySkuListFinishedListener _querySkuListFinishedListener =
new IabHelper.QuerySkuListFinishedListener() {
    public void onQuerySkuListFinished(IabResult result, List<SkuDetails>
        skuDetails, String type) {
        ...
        UnityPlayer.UnitySendMessage(EVENT_MANAGER,
            QUERY_SKU_LIST_SUCCEEDED_CALLBACK, jsonSkuDetails);
    }
};
```

- ii. The list message is received by `OnQuerySkuListSucceeded()`. This callback lists the SKUs retrieved from the app store

```
Shop.cs
// Listener callback if an SKU list query is successful

private void OnQuerySkuListSucceeded(List<SkuDetails> skuDetails)
{
    if (!infoCanvas.activeSelf) {
        infoCanvas.SetActive (true);
        infoLog.text = "SKU List Query Success!";
    }

    string logString = "SKU List:", menuString = "Available SKU:\n";
    for (int i=0; i<skuDetails.Count; i++) {
        logString += " "+skuDetails[i];
        menuString += i+1 + ".) " + skuDetails[i] + "\n";
    }
    shopText.text = menuString;
}
```

- b. When the list generation query fails, `OnQuerySkuListFailed()` is invoked.

```
Shop.cs
// Listener callback if an SKU list query fails

private void OnQuerySkuListFailed(string error)
{
}
```

```

        if (!infoCanvas.activeSelf) {
            infoCanvas.SetActive (true);
            infoLog.text = error;
        }
    }
}

```

3.4.2.2. Generating and Getting a List of Purchased Items

Service apps can generate and get a list of items the current user has purchased from Samsung servers.

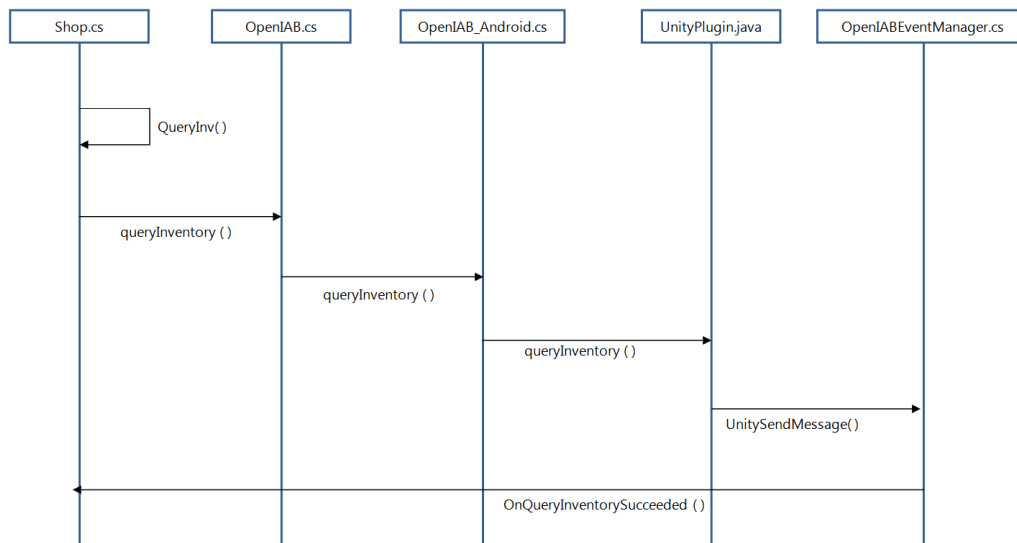
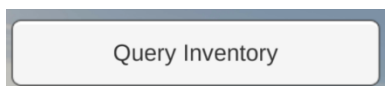


Figure 4 API Call Sequence: Generating and Getting a List of User-Purchased Items

To generate and get a list of user-purchased items:

1. In the City Flyer UI, click **Query Inventory**



2. The button click invokes the function QueryInv() of Shop.cs, which calls OpenIAB.queryInventory() and passes the skuList parameter. The Query Inventory function gets the list directly from the current app store which generates and returns the list.

```

Shop.cs
// Called on the onClick() method of the Query Inventory button
// Starts the process to display a list of items purchased to-date

public void QueryInv(){
    OpenIAB.queryInventory(skuList);
}

```


3. `OpenIAB.queryInventory()` calls the `queryInventory()` function of `OpenIAB_Android.cs` for Android devices and passes the `skuList` in the `extraParams` parameter.

```
OpenIAB.cs
/**
 * Sends a request to get all completed purchases
 * Added extraParams parameter to match function signature
 * @param extraParams this allows additional parameters if needed.
 * Function will still work in legacy code. */

public static void queryInventory(params object[] extraParams){
    _billing.queryInventory(extraParams);
}
```

4. `OpenIAB_Android.queryInventory` uses a Java Native Interface (JNI) to access the `queryInventory()` method in the `UnityPlugin.java` file.

```
OpenIAB_Android.cs
/**
 * Added extraParams parameter to match interface
 * @param extraParams this allows additional parameters if needed.
 * Function will still work with legacy code.
 */

public void queryInventory(params object[] extraParams){
    ...
    methodId = AndroidJNI.GetMethodID(_plugin.GetRawClass(), "queryInventory",
        "([Ljava/lang/String;)V");
    AndroidJNI.CallVoidMethod(_plugin.GetRawObject(), methodId, args);
}
```

5. `UnityPlugin.java` accesses the library-level classes that provide the functions for querying the inventories. After the inventory query processing is completed, `onQueryInventoryFinished()` is invoked.

- a. When the list generation query is successful:
 - i. `onQueryInventoryFinished()` sends the message with the list back to `Shop.cs`.

```
UnityPlugin.java
public void queryInventory(final Object... extraParams){
    UnityPlayer.currentActivity.runOnUiThread(new Runnable() {
        @Override
        public void run() {

            _helper.queryInventoryAsync(_queryInventoryListener, extraParams);
        }
    });
}

IabHelper.QueryInventoryFinishedListener _queryInventoryListener = new
IabHelper.QueryInventoryFinishedListener() {
    public void onQueryInventoryFinished(IabResult result, Inventory
inventory) {
        ...
    }
}
```

```

        UnityPlayer.UnitySendMessage(EVENT_MANAGER,
        QUERY_INVENTORY_SUCCEEDED_CALLBACK, jsonInventory);
    }
};

```

- ii. This message is received by `OnQueryInventorySucceeded`, which displays the list.

```

Shop.cs
// Listener callback if an inventory query is successful
private void OnQueryInventorySucceeded(Inventory inventory){
    if (!infoCanvas.activeSelf){
        infoCanvas.SetActive (true);
    }
    string item = "";
    foreach(string pchase in inventory.GetAllOwnedSkus()){
        item += pchase+"\n";
    }
    inventoryText.text = "Items bought: \n" + item;
}

```

- b. When the list generation query fails, `OnQueryInventoryFailed()` is invoked.

```

Shop.cs
// Listener callback if an inventory query fails
private void OnQueryInventoryFailed(string error)
{
    if (!infoCanvas.activeSelf) {
        infoCanvas.SetActive (true);
        infoLog.text = error;
    }
}

```

3.4.2.3. Getting the Previously Generated List of Purchased Items

Service apps can get a previously generated list of items the current user has purchased from the cache.

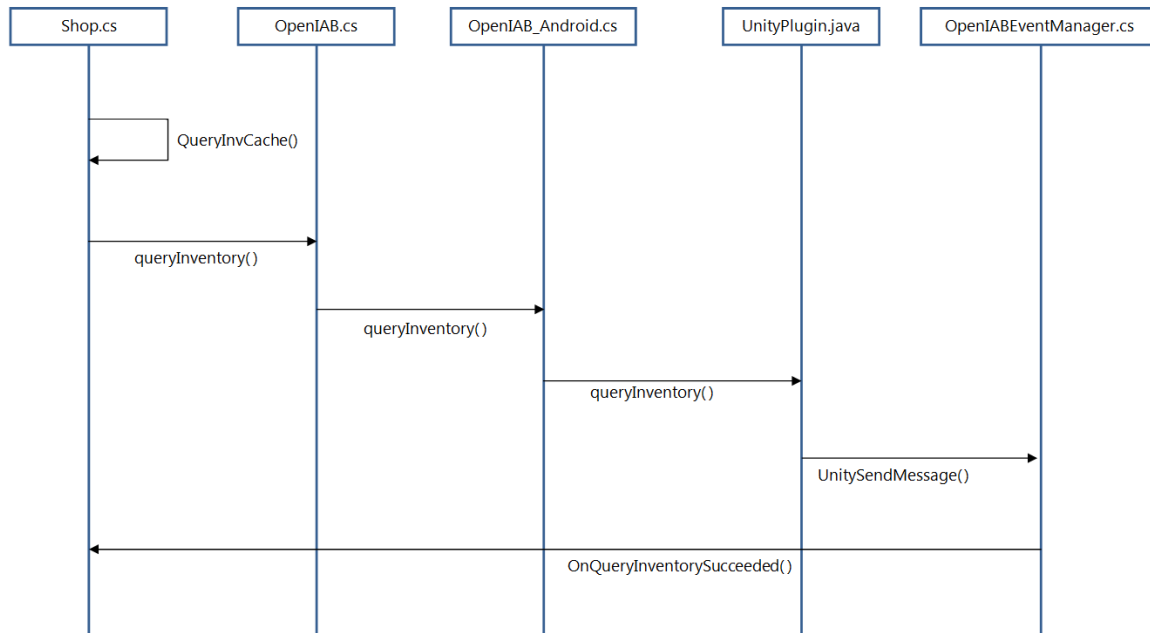
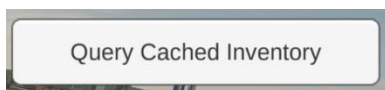


Figure 5 API Call Sequence: Getting the Cached List of User-Purchased Items

To get the locally cached list of user-purchased items:

1. In the City Flyer UI, click **Query Cached Inventory**



2. The button click invokes QueryInvCache(), which calls OpenIAB.queryInventory() with an additional parameter "true" which gets the cached list of items based on the skuList parameter.

```
Shop.cs
// Called on the onClick() method of the Query Cached Inventory button
// Starts the process to display a list of items currently bought from the
cache
public void QueryInvCache(){
    OpenIAB.queryInventory(skuList, "true");
}
```

The rest of the process follows the same sequence and callbacks as the **Query Inventory** feature (To generate and get a list of user-purchased items: steps 3. through 5.) EXCEPT that items in the local cache are searched and retrieved.

3.4.3. Purchasing Commercial Items

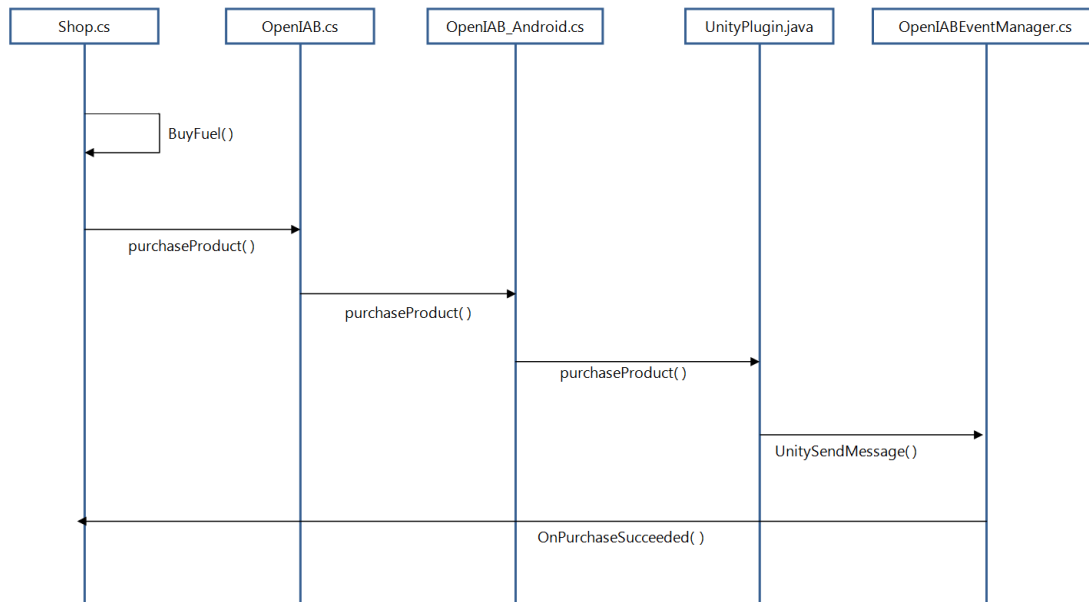


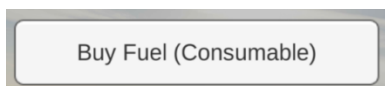
Figure 6 API Call Sequence: Purchasing Commercial Items

3.4.3.1. Purchasing Consumable Items

The Buy Fuel feature of this application is of consumable type. Once the fuel runs out, you will need to repurchase. Multiple purchases can be done with this type of commercial item.

To purchase a consumable item:

1. In the City Flyer UI, click **Buy Fuel**



2. The button click invokes the function BuyFuel() of Shop.cs, which calls OpenIAB.purchase Product() and passes the SKU of the item (SKU_PLANE_FUEL).

Shop.cs

```
// Called on the onClick() method of the Buy Fuel button
// Starts the purchase flow for fuel
public void BuyFuel() {
    OpenIAB.purchaseProduct(SKU_PLANE_FUEL);
}
```

3. `OpenIAB.purchaseProduct()` calls the `purchaseProduct()` function of `OpenIAB_Android.cs` for Android devices.

In this call and others, the item SKU is passed as the parameter `sku`. The parameter `developerPayload` is assigned a blank string because the developer payload feature is not supported by Samsung IAP.

```
OpenIAB.cs
/**
 * Purchases the product with the given sku and developerPayload
 * @param product ID
 * @param developerPayload payload to verify transaction
 */
public static void purchaseProduct(string sku, string developerPayload = ""){
    billing.purchaseProduct(sku, developerPayload);
}
```

4. `OpenIAB_Android.purchaseProduct` calls `purchaseProduct()` of `UnityPlugin.java` via the method `Call()`.

```
OpenIAB_Android.cs
public void purchaseProduct(string sku, string developerPayload = ""){
    ...
    plugin.Call("purchaseProduct", sku, developerPayload);
}
```

5. `UnityPlugin.java` accesses the library-level classes that provide the functions for buying the items. After the item purchase processing is completed, `onIabPurchaseFinished()` is invoked.
 - a. When the item purchase process is successful:
 - i. `onIabPurchaseFinished()` sends the message back to `Shop.cs`.

```
UnityPlugin.java
/**
 * Callback for when a purchase is finished
 */
IabHelper.OnIabPurchaseFinishedListener _purchaseFinishedListener = new
IabHelper.OnIabPurchaseFinishedListener() {
    public void onIabPurchaseFinished(IabResult result, Purchase
purchase) {
        ...
        UnityPlayer.UnitySendMessage(EVENT_MANAGER,
                                    PURCHASE_SUCCEEDED_CALLBACK, jsonPurchase);
    }
};
```

- ii. This message is received by `OnPurchaseSucceeded`. It performs the corresponding action based on the item purchased by the user. To learn more about the `Purchase` object, see the *OpenIAB API Reference*.

```
Shop.cs
// Listener callback if the purchase is successful
private void OnPurchaseSucceeded(Purchase purchase) {
    // Check what was purchased and update the game
    switch (purchase.Sku) {
        case SKU_PLANE_FUEL:
            FuelController.fuelAmount += 25;
            if (!infoCanvas.activeSelf) {
                infoCanvas.SetActive (true);
                infoLog.text = "Added 25 Fuel!";
            }
            break;
    }
}
```

- b. When the item purchase process is fails:

- i. `OnPurchaseFailed()` is invoked.
For error response code details, see the *OpenIAB API Reference*.

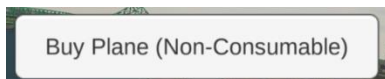
```
Shop.cs
// Listener callback if the purchase fails
private void OnPurchaseFailed(int errorCode, string error)
{
    if (!infoCanvas.activeSelf) {
        infoCanvas.SetActive (true);
        infoLog.text = error;
    }
}
```

3.4.3.2. Purchasing Non-Consumable Items

The Buy plane feature of this application is of type non-consumable. This means that this item can be purchased only once. The purchased plane does not expire and can be re-used an unlimited number of times.

To purchase a non-consumable item:

1. In the City Flyer UI, click **Buy Plane**



2. The button click invokes the function `BuyPlane()` of `Shop.cs`, which calls `OpenIAB.purchase Product()` and passes the SKU of the item (`SKU_CHANGE_PLANE`).

```
Shop.cs
// Listener callback if the purchase is successful
```

```
private void OnPurchaseSucceeded(Purchase purchase) {
    switch (purchase.Sku) { // Check what was purchased and update the game
        case SKU_CHANGE_PLANE:
            if (!infoCanvas.activeSelf) {
                infoCanvas.SetActive (true);
                infoLog.text = "Plane changed to F-16!";
            }
            oldPlane.SetActive (false);
            newPlane[0].SetActive (true);
            break;
    }
}

Shop.cs
// Called on the onClick() method of the Buy Plane button
// Starts the purchase flow for a new airplane
public void BuyPlane() {
    OpenIAB.purchaseProduct (SKU_CHANGE_PLANE);
}
```

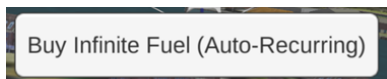
Plane purchases follow the same sequence as **Buy fuel**, starting with the function `BuyPlane()`, and ending with case `SKU_CHANGE_PLANE` in `OnPurchaseSucceeded()`.

3.4.3.3. Purchasing Auto-Recurring Subscription Items

The Buy infinite fuel feature of this application is of type auto-recurring. This means that the subscription for infinite fuel is automatically renewed and repurchased after every specified period. The payment cycle and free trial period can be specified when you add the item in the Samsung Seller site.

To purchase an auto-recurring subscription item:

1. In the City Flyer UI, click **Buy Infinite Fuel**



2. The button click invokes the function `InfiniteGas()` of `Shop.cs`, which calls `OpenIAB.purchase Product()` and passes the SKU of the item (`SKU_INFINITE_FUEL`).

```
Shop.cs
// Called on the onClick() method of the Buy Infinite Gas button
// Starts the purchase flow for infinite gas
public void InfiniteGas() {
    OpenIAB.purchaseSubscription (SKU_INFINITE_FUEL);
}
```

```

Shop.cs
// Listener callback if the purchase is successful
private void OnPurchaseSucceeded(Purchase purchase){
    // Check what was purchased and update the game
    switch (purchase.Sku){
        case SKU_INFINITE_FUEL:
            if (!infoCanvas.activeSelf){
                infoCanvas.SetActive (true);
                infoLog.text = "Infinite Fuel Activated!";
            }
            FuelController.fuelAmount = 100;
            FuelController.usage = 0;
            break;
    }
}

```

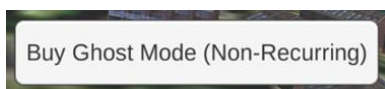
Infinite fuel purchases follow the same sequence as Buy fuel, starting with the function `InfiniteGas()`, and ending with case `SKU_INFINITE_FUEL` in `OnPurchaseSucceeded()`.

3.4.3.4. Purchasing Non-Recurring Subscription Items

The Buy ghost mode feature of this application is of type non-recurring. This means that this item can be purchased after a specified expiry period. The duration can be specified when you add the item in the Samsung Seller site.

To purchase a non-recurring subscription item:

1. In the City Flyer UI, click **Buy Ghost Mode**



2. The button click invokes the function `GhostMode()` of `Shop.cs`, which calls `OpenIAB.purchase Product()` and passes the SKU of the item (`SKU_GHOST_MODE`).

```

Shop.cs
// Called on the onClick() method of the Ghost Mode button
// Starts the purchase flow for ghost mode
public void GhostMode(){
    OpenIAB.purchaseSubscription(SKU_GHOST_MODE);
}

```


Shop.cs

```
// Listener callback if the purchase is successful
private void OnPurchaseSucceeded(Purchase purchase){
    // Check what was purchased and update the game
    case SKU_GHOST_MODE:
        if (!infoCanvas.activeSelf) {
            infoCanvas.SetActive (true);
            infoLog.text = "Ghost Mode Active!";
        }
        isGhostModeEnabled = true;
        break;
    }
}
```

Ghost mode purchases follow the same sequence as **Buy fuel**, starting with the function `GhostMode()`, and ending with case `SKU_GHOST_MODE` in `OnPurchaseSucceeded()`.

3.5. Terminating Commercial Item Processing

In order to release the resources used by the OpenIAB billing service and event listeners, the following routines need to be added to your Unity project source code.

3.5.1. Unsubscribe to Plugin Events

To prevent memory leaks or to free memory, unsubscribe to OpenIAB plugin events via the `OpenIABEventManager` class, which unregisters the listeners that were previously set in `Awake()`. In *City Flyer*, unsubscribing is implemented in the `onDestroy()` class. In Unity, `onDestroy()` is invoked when the class will be destroyed.

Shop.cs

```
// Remove listeners when the application exits
private void OnDestroy()
{
    // Remove listeners from the event manager to avoid nasty leaks
    OpenIABEventManager.billingSupportedEvent -= OnBillingSupported;
    OpenIABEventManager.billingNotSupportedEvent -= OnBillingNotSupported;
    OpenIABEventManager.queryInventorySucceededEvent -=
OnQueryInventorySucceeded;
    OpenIABEventManager.queryInventoryFailedEvent -= OnQueryInventoryFailed;
    OpenIABEventManager.purchaseSucceededEvent -= OnPurchaseSucceeded;
    OpenIABEventManager.purchaseFailedEvent -= OnPurchaseFailed;
    OpenIABEventManager.consumePurchaseSucceededEvent -=
OnConsumePurchaseSucceeded;
    OpenIABEventManager.consumePurchaseFailedEvent -= OnConsumePurchaseFailed;
    OpenIABEventManager.transactionRestoredEvent -= OnTransactionRestored;
    OpenIABEventManager.restoreSucceededEvent -= OnRestoreSucceeded;
    OpenIABEventManager.restoreFailedEvent -= OnRestoreFailed;
}
```

3.5.2. Unbind from the IAP Billing Service

After working with the in-app purchases, unbind from the IAP billing service by calling the `unbindService()` method, which unbinds and shuts down the billing service. This unbinding is implemented in the `onDestroy()` method in the `Shop.cs` file.

Shop.cs

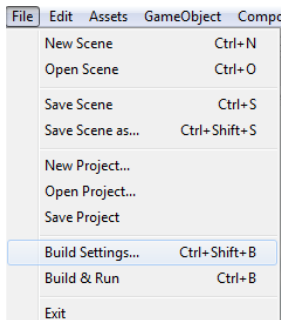
```
// Remove listeners when the application exits
private void OnDestroy()
{
    ...
    // Unbind from the IAP billing service
    OpenIAB.unbindService();
}
```

3.6. Building Your Service App

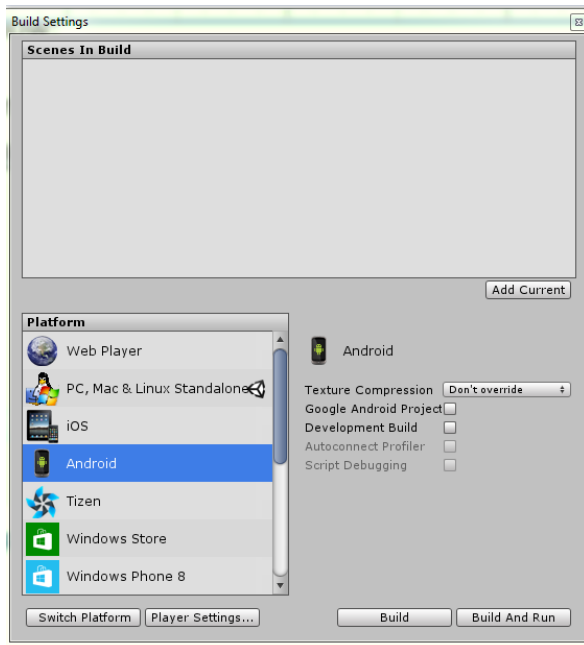
After successfully integrating the OpenIAB plugin to your Unity project, you are now ready to build your app.

To build your Unity service app:

1. Start up Unity.
2. Click **File > Build Settings**



3. In the Build Settings window:



- a. Select **Platform: (Android)**
- b. Click **Build**

3.7. Testing Your Service App Integration

In order to verify if the plugin is successfully integrated to your Unity project, the information below about the operating modes of your service app and the common plugin errors is necessary.

3.7.1. Service App Operating Mode

Your service app operations can be switched between normal, commercial production mode, and development modes for testing using the `samsungIapMode(int mode)`.

For operating mode details, see section A.1 of this document.

You can force set the app store option to Samsung Galaxy Apps Store using the `storeModeSamsung(boolean enabled)`. For more details about the values for these methods, see section 1.1 of the *Open IAB Unity Plugin API Reference*.

```
OpenIAB.samsungIapMode(OptionsController.samsungMode);  
OpenIAB.storeModeSamsung(OptionsController.isSamsungForced);
```

3.7.2. OpenIAB Plugin Error Returns and Integration Errors

IAP_ERROR_PRODUCT_DOES_NOT_EXIST	Description:	Encountered when the SKU mapping does not match the <code>itemID</code> or <code>itemGroupID</code> .
	Solution:	Ensure that the <code>itemID</code> or <code>itemGroupID</code> matches the SKU mapping.
		The <code>itemID</code> and <code>itemGroupID</code> are located at the Samsung Apps Seller Office website.
		NOTE: The <code>itemGroupID</code> (12-digit numeric string) can be omitted, but if you add it, note that it is unique to every service app.
IAP_ERROR_ALREADY_PURCHASED	Description:	Encountered when a non-consumable, non-recurring subscription, or auto-recurring subscription item being purchased has already been bought
	Solution:	Ensure that service app logic and code prevents users purchasing non-consumable, non-recurring subscription, or auto-recurring subscription items that have already been bought.

Incorrect app store is chosen	<p>Description:</p> <p>Encountered when the billing store is incorrectly set during sideloading via ADB.</p> <p>Solution:</p> <p>Force the service app to select Samsung IAP when it is set in conjunction with the BEST_FIT mode regardless of the options set during side loading.</p> <p>This solution was applied in the sample service app in the OpenIAB Plugin Integration section to ensure that it selects Samsung IAP, whether or not a billing store was specified during sideloading.</p>
Nothing happens after purchase or query	<p>Description: Purchases and queries launch events after they execute. These events are defined in the <code>OpenIABEventManager.prefab</code> file (e.g., <code>OnBillingSupported</code>, <code>OnPurchaseSucceeded</code>). Nothing will happen after purchases or queries if this is not attached to your project.</p> <p>Solution: Attach the <code>OpenIABEventManager.prefab</code> file to your project. Refer to the first step in configuring the OpenIAB Unity plugin under the OpenIAB Plugin Integration section.</p>
NullPointerException: IAPConnector is null	<p>Description: This error is due to not initializing the <code>IAPConnector</code>. Without initializing this, it will be null and consequently, the events that need it will be using a null object.</p> <p>Solution: Initialize OpenIAB during startup using the <code>OpenIAB.init()</code> function. This will initialize the <code>IAPConnector</code> as well.</p>
Billing fails during setup	<p>Description: No requested OpenIAB plugin API method calls based on Samsung IAP functionality are executed because Samsung IAP Client and/or Samsung Billing Client are NOT installed on the test device.</p> <p>Solution: Ensure that Samsung IAP Client and Samsung Billing Client are installed on the test device.</p>

A. Integration Interdependencies

This section describes aspects of OpenIAB plugin integration that interact with other aspects, and/or can affect multiple integration steps.

A.1. Item Purchase Processing Modes

`OpenIAB.samsungIapMode(int MODE)` supports 3 Samsung IAP modes that control how requested commercial item purchases are processed. To test the service under various conditions for production and testing. During development, it is recommended to test your service app under both test modes.

In developer test mode, each consumable item can be re-purchased at any time, each non-consumable item can be re-purchased 10 minutes after the previous purchase, each non-recurring item can be repurchased after their specified time of expiry, and each auto-recurring subscription item can be re-purchased 2 hours after the previous purchase.

CAUTION: Before releasing your service app for app store distribution, set it to production mode.
In either test mode, NO financial transactions will occur after any item purchase request.

Production Mode `IAP_MODE_COMMERCIAL` 0

This is the mode for production where the actual payment process occurs. When releasing your application, make sure to set it to production mode. The parameter value is 0.

```
OpenIAB.samsungIapMode(IAP_MODE_COMMERCIAL); //Production Mode
```

Test Success Mode `IAP_MODE_TEST_SUCCESS` 1

This test mode always returns successful results. The parameter value is 1.

```
OpenIAB.samsungIapMode(IAP_MODE_TEST_SUCCESS); //Developer Test Mode (Success)
```

Test Failure Mode `IAP_MODE_TEST_FAIL` -1

This test mode always returns failed results. The parameter value is -1.

```
OpenIAB.samsungIapMode(IAP_MODE_TEST_FAIL); //Developer Test Mode (Failure)
```

A.2. Server-to-Server Purchase Verification

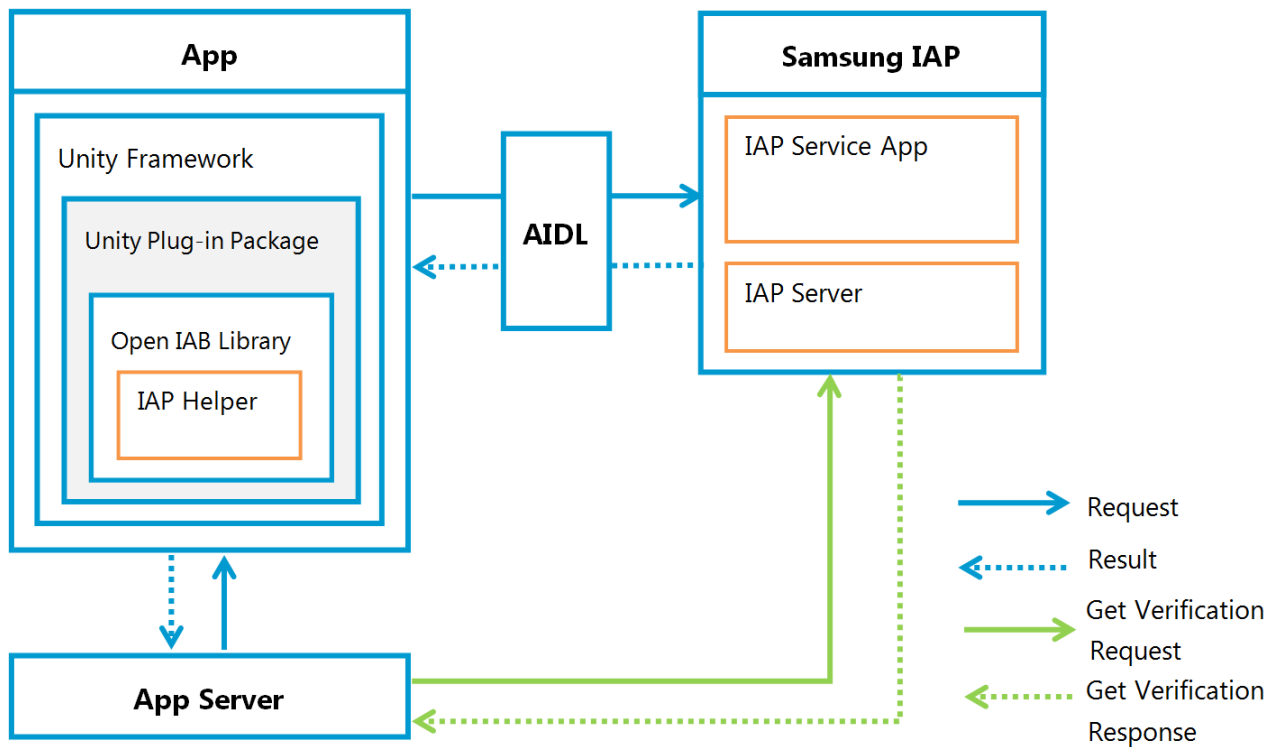


Figure 7 Server-to-Server Verification

After a user makes a purchase, the developers can determine whether or not the purchase was successfully processed by Samsung Billing, using the purchase token to visit a URL. Purchase verification is typically performed by the service app server (which is explained below). However, purchase verification can be performed directly on the service app.

To perform server-to-server purchase verification:

1. Use the verify URL:

```
https://iap.samsungapps.com/iap/appsItemVerifyIAPReceipt.as?protocolVersion=2.0
```

2. Get the purchaseID. You can get the purchaseID from the:

- **onPurchaseSucceeded callback (Unity Level)**

This is a Unity level approach. In the Shop.cs file, get the token attribute from the object **Purchase** purchase by using `purchase.Token` as shown below:

```
Shop.cs
private void OnPurchaseSucceeded(Purchase purchase) {
    ...
    String purchaseID = purchase.Token;
    ...
}
```

- **Purchase finished logs**

After a purchase, get the token included in the purchase finished logs. This can be enabled and disabled in the source code using `enableDebugLogging()`. Refer to section 1.1.5 of the OpenIAB Unity Plugin API Reference. For example:

```
D      24718      OpenIAB-UnityPlugin  Purchase finished: IabResult: 0, 0:OK,
purchase: PurchaseInfo(type:inapp):
{"orderId":TPMTID20151104PHI2037906,"packageName":com.samsung.bostoniab,"pr
oductId":sku_plane_fuel,"purchaseTime":1446604114390,"purchaseState":0,"dev
eloperPayload":,"token":9954cec99a965fef6490b3fec3baf9ba7d98335a6d5b1c66377
4c6503d442ee7}
```

For production apps, it is recommended to disable debug logging.

3. Append the purchaseID to the URL following the format below:

```
&purchaseID=<token>
```

4. The following is an example that uses the forma specified above:

```
https://iap.samsungapps.com/iap/appsItemVerifyIAPReceipt.as?protocolVersion=2.
0&purchaseID=9954cec99a965fef6490b3fec3baf9ba7d98335a6d5b1c663774c6503d442ee7
```

5. Verify URL

6. The link will provide the results of the verification in JSON format.

- a. If the purchase is valid:

```
{
  "itemId" : "57515",
  "paymentId":"ZPMTID20131122GBI0015292",
  "itemName":"Test Pack",
  "itemDesc":"IAP Test Item. Best value!",
  "purchaseDate":"2013-11-22 04:22:36",
  "paymentAmount":"9.000",
  "status":"true",
  "paymentMethod":"CreditCard",
  "mode":"REAL",
}
```

- b. If the purchase is invalid:

```
{"status":"false"}
```

For more details about the description of result values, see section 4 of the Samsung IAP v3.0 Programming Guide.

B. Service App and Item Registration, Certification, and Management

During development before you configure the OpenIAB plugin, initially register your service and its commercial items with Samsung Galaxy Apps Seller Office. This will enable you to test your item purchase integration.

After proper OpenIAB Plugin Integration and production environment configuration (see below), your service app and its commercial items must be: configured for normal, commercial production, their Seller Office registration must be updated, and they must be certified by Samsung in order for them to be distributed via the Samsung Galaxy Apps Store and managed via the Samsung Galaxy Apps Seller Office.

To initially register your development service app and items with Seller Office:

NOTE: You must initially register your service app and its items during development. You can change uploaded files and most entries after initial registration at any time. Entries that cannot be changed are indicated in the procedure. Required field entries are indicated in the Seller Office UI and in the procedure.

- Get the following (if applicable):
 - Development service app binary file(s)
 - Development commercial item information
 - Information needed to make appropriate field entries
- 1. Follow the procedure **To register a service app and its commercial items:** in the *Samsung Seller Office Service Application Registration Guide*.

To configure your service app for the commercial production environment:

1. Set your service app to production mode before releasing it for app store distribution. This can be done by setting the Samsung IAP mode to 0.

```
OpenIAB.samsungIapMode(0); //Production Mode
```

For more details about item purchase processing modes, see section A.1.

To register your production service app and items with Seller Office:

NOTE: You must configure your service app and its items for commercial production and perform this procedure and complete registration prior to certification for distribution.

- Get the following (if applicable):
 - Commercial production service app binary file(s)
 - Commercial production item information
 - Information needed to make appropriate field entries
- 1. Follow the procedure **To register a service app and its commercial items:** in the *Samsung Seller Office Service Application Registration Guide*.

Commonly encountered problems when registering commercial items include:

- No binary file(s) have been uploaded.
- The binary does not support IAP. The following permission that allows the service app to support IAP has not yet been added to the Android manifest file:

```
<uses-permission android:name="com.sec.android.iap.permission.BILLING" />
```

- For more details on setting Samsung IAP privileges, see section 2.3 of the Samsung IAP v3.0 Programming Guide.

To certify your production service app and items:

NOTE: You must complete registration of your production service app and its items prior to certification for distribution.

- Get the following (if applicable):
 - Commercial production service app binary file(s)
 - Commercial production item information
 - Information needed to make appropriate field entries
- 1. Follow the procedure **To submit a registered service app for certification:** in the Samsung Seller Office Service Application Certification Guide.

To manage your distributed service app and items:

NOTE: Your service app and its items must pass Samsung certification for distribution.

1. Follow the applicable procedures in the *Samsung Seller Office Service Application Management Guide*.

C. App Store Selection for Item Purchase

The following logic is used to determine the app store from which to purchase an item.

To determine the app store that an item is to be purchased from:

1. Store Key Filtering

Filters the list of all supported app stores as specified.

- a) When `verifyMode=VERIFY_SKIP`, this step is NOT performed.
- b) From the list of all supported app stores, different processes are used to retain or remove app stores.
 - All app stores that generate and use store keys (RSA keys) are retained.
 - For app stores that do not generate and use store keys:
 - For the app stores specified in `availableStoreNames`, `availableStores`, or `preferredStores`:
 - When `verifyMode=VERIFY EVERYTHING`, an exception is thrown for each app store.
OR
 - When `verifyMode` is NOT `VERIFY EVERYTHING`, each app store is removed.
 - App stores that check the permission in the Android manifest.
 - App stores that check the classes required.
 - App stores (such as Samsung) that check if the activity is not null.
 - All other app stores that meet the required dependencies are retained.
 - For app stores that do not meet the required dependencies, an exception is thrown for the app stores specified in `availableStoreNames`, `availableStores`, or `preferredStores`.

2. Adding Developer-Specified App Stores

Ensures the list of remaining app stores contains the app stores specified in `availableStores` and `availableStoreNames`. If necessary the specified stores are added.

3. storeSearchStrategy Filtering

Filters the list of remaining app stores based on the method specified by `storeSearchStrategy`:

SEARCH_STRATEGY_BEST_FIT: `storeSearchStrategy` filtering is NOT performed.
Go to step 4. checkInventory App Store Determination.

SEARCH_STRATEGY_INSTALLER: Perform the following filtering:

- If the package name of the service app installer is valid, retain the app store that the service app was downloaded from in the list of remaining app stores.
- If the package name of the service app installer is NOT valid, remove the app store.

When 1 or more app stores remain, go to step 4. checkInventory App Store Determination.

When 0 app stores remain, the purchase CANNOT be made because no app stores remain in the list.
(The store selection process stops.)

SEARCH_STRATEGY_INSTALLER_THEN_BEST_FIT:

Perform the following filtering:

- If the package name of the service app installer is valid, retain the app store that the service app was downloaded from in the list of remaining app stores.
- If the package name of the service app installer is NOT valid, remove the app store.

Go to step 4. checkInventory App Store Determination, regardless of how many app stores remain in the list (1 or more).

4. checkInventory App Store Determination

Determines the app store to be used or that the purchase cannot be completed, based on the method specified by checkInventory:

- | | |
|--------------|---|
| TRUE | Use the app store that the user most recently purchased the item from. |
| FALSE | Sequentially check the list of remaining app stores for available billing, EITHER: <ul style="list-style-type: none">• Use the first app store with available billing.OR• The purchase CANNOT be made because there are no app stores in the list with available billing. |