

Simple IAP System

Documentation

V3.1

Scripting Reference	1
1 Getting Started	2
2 Creating In-App Products in Unity	3
3 Instantiating Products in the Shop	4
4 Customizing IAP Item prefabs	5
5 Handling IAP Callbacks	6
6 Encrypting device storage	7
7 Shop Templates explained	8
8 Receipt Verification	9
9 Adjusting In Game Content remotely	10
10 Localizing IAP data	10
11 Contact	11

Thank you for buying Simple IAP System!
Your support is greatly appreciated.

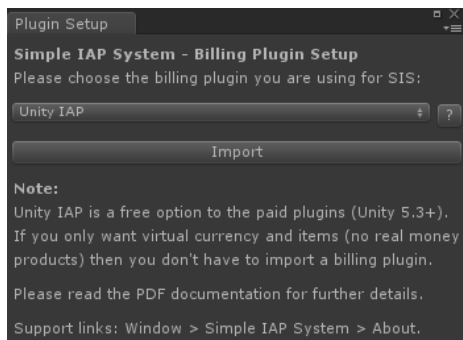
Scripting Reference


www.rebound-games.com/docs/sis

1 Getting Started

For video tutorials, please visit our [YouTube channel](#).

- 1 Open our billing plugin setup window under Window > Simple IAP System > Plugin Setup and import our customized billing package you would like to use with SIS. After this step you could see some errors in the console, until you proceed to step 2.



- 2 Billing Plugin - specific instructions:
 - **Unity IAP:** Running Unity 5.3+, open the  services tab. Enable your project for Unity In-App Purchasing and import Unity IAP's billing plugins (in the same window).
 - **Stan's Assets:** Import your Android/iOS Native plugins or Ultimate Mobile package from Stans Assets. Do not specify any products in its plugin billing settings.
 - **Prime31:** Import your Android/iOS billing plugin or the combo package from Prime31.
 - **VoxelBuster:** Import your Android and/or iOS billing plugin or Cross Platform Ultra Pack from Voxel Busters. Do not specify any products in its plugin billing settings.

You don't have to import a billing plugin if you don't want to charge your users, and only need virtual products and currency – such as buying a sword with coins earned in-game.

- 3 Drag the "IAP Manager" prefab from SIS > Prefabs > Resources into the very first scene of your game. Start from the 'AllSelection' scene if you want to run the example scenes of Simple IAP System (the IAP Manager is already in there).
- 4 Drag the "Shop Manager" prefab from SIS > Prefabs into the scene where you want to display in-app purchases (your own shop scene, or use one of the templates included).
- 5 **Optional:** If this is the first time you get in touch with mobile development and in-app purchases, please have a look at our store-specific guides on [the forums](#) for successful configurations on App Stores, and decide if you need them.

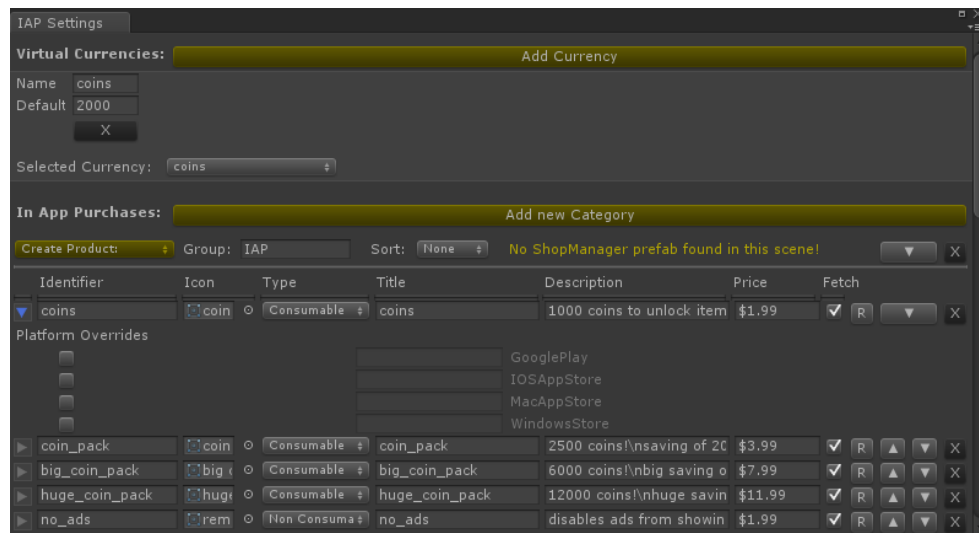
2 Creating In-App Products in Unity

Before linking in-app products for real money in Unity, you have to create them in the App Store (Google Play, iTunes Connect, etc.). If you didn't do this yet, please see step 5 on the last page.

In Unity, the IAP Settings editor is your main spot for managing IAPs, be it for real or virtual money.



Please open it by navigating to Window > Simple IAP System > IAP Settings.



In App Purchases: here you specify your IAPs for real money (must match your App Stores ids). If you have different identifiers for the same product per store, you can use platform overrides.

Virtual Economy: here you create IAPs for virtual currency, depending on your game logic.

These are the variables which need to be defined in the editor:

- **Group:** unique name of the group you are adding products to
- **Identifier:** your unique product identifier (or platform-specific id, see above)
- **Icon:** the sprite texture you want to use as the icon
- **Type:** product type in the billing model
 - Consumables: can be purchased multiple times, for real money/currency (e.g. coins)
 - Non-consumables: one time purchase for real money/currency (e.g. bonus content)
 - Subscriptions: periodically bills the player, for real money (e.g. service-based content)
- **Fetch:** whether local product data should be overwritten by fetched App Store data
- **Title, Description:** descriptive text for the product in your shop
- **Price:** product price (string value for real money, int value for virtual currency). Virtual products will start as pre-purchased if their price is zero.
- **R:** opens the requirement window. The user has to meet this requirement in order to unlock the product first. See the 'Customizing IAP Item prefabs' section for more details.

Exception: **'restore'**. For restoring real money purchases in your store, simply add a product with the id 'restore', set its type to Consumable and keep 'Fetch' unchecked. The IAP Manager will recognize this id and call the appropriate restore methods for you.

Apple will reject your app if you don't provide a restore button for your in-app purchases.

For virtual products, you will have to specify one or more virtual currencies first. Set a name and default value the user should start with. It is not recommended to rename/remove currencies in production, as this could lead to inconsistency and loss in funds.

Side note: always keep the IAP Manager prefab in the Resources folder. When making changes to the IAP Settings editor, these changes are automatically saved to the prefab. Thus, if you are upgrading to a new version of SIS, make sure to keep a local copy of your IAP Manager prefab.

3 Instantiating Products in the Shop

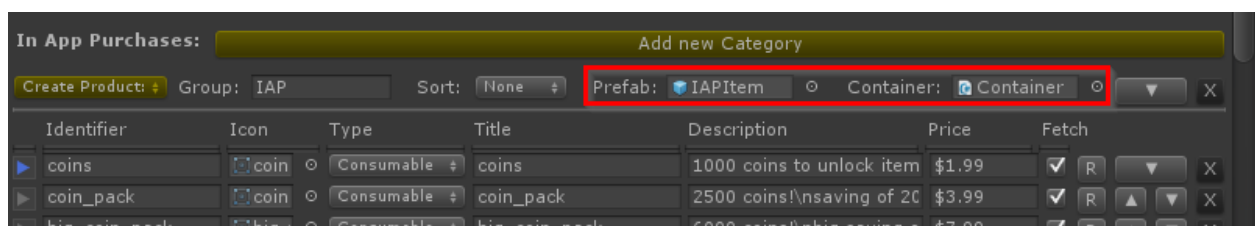
The Shop Manager prefab in your shop scene will do all the work for you. Let's have a look at it.



Please open one of the example shop scenes, e.g. *SIS > Scenes > Vertical*

For each group in the IAP Settings editor, you only have to define what visual representation (**prefab**) and where (**container**) you want to instantiate your products. There are several predefined IAP Item prefabs to choose from, located in the *SIS > Prefabs > Vertical/Horizontal* folder. When you play the scene, the Shop Manager instantiates this prefab for each item, parented to the container, and Unity's GridLayoutGroup component aligns them nicely in the scene.

Your container transform in the scene needs the IAPContainer script attached to it.



After instantiating your products as items in the scene, the Shop Manager also makes sure to set it to the correct state. This means that the shop item for a purchased product does not show the buy button anymore, or an equipped item has a button to deselect it again and so on.

As public variables, the Shop Manager exposes references to a game object and a Text, which are being used for showing the feedback window in case purchases succeed or fail.

4 Customizing IAP Item prefabs

As mentioned in the last chapter, IAP Item prefabs visualize IAP products in your shop at runtime. These prefabs have an IAP Item component attached to them, that has references to every important aspect of the item, e.g. to descriptive labels, the buy button, icon texture and so forth. Based on the product's state, IAP Items show or hide different portions of their prefab instance.



For variable descriptions, please refer to the [IAPItem](#). The following item states are supported:

Default (initial state)



Purchased (user owns this product)



Single Select (unequips others)



Multi Select (does not unequip others)

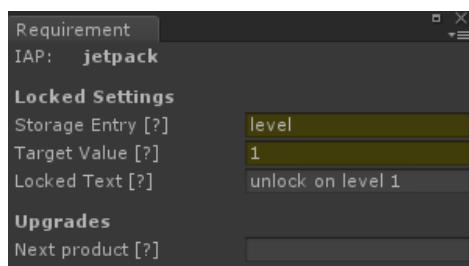


Locked (requirement not met)



Possible states for your item are specified by assigning references to the IAP Item component:

- **Default:** the item needs at least descriptive labels (title/description, price) and a buy button
- **Purchased:** assign a game object to the 'Sold' slot, which gets activated on sold products
- **Single Select:** assign the 'Select Button', but leave the 'Deselect Button' empty
- **Multi Select:** assign both 'Select Button' and 'Deselect Button'
- **Locked:** prepare your prefab for the locked state first by disabling the buy button, descriptive labels etc. and showing the 'Locked Label' and 'Hide On Unlock' game objects. If the item gets unlocked, it hides 'Hide On Unlock' and shows 'Show On Unlock' instead.



This is the requirement for the jetpack product, defined in the IAP Settings editor. Hover over the labels to see what they mean. For example, here we unlock the jetpack if the player reached level 1.

The vertical/horizontal menu scenes showcase this product in the Custom sub menu.

Upgrades (multiple levels for one product) can be specified here too. Simply enter the product id that comes after this one, and the shop item gets replaced with the next level after the user bought it. Your upgrades do not need to be instantiated in the scene, thus you can leave their Prefab & Container slot empty. The "speed" product is a sample, showcased in the Items section.

5 Handling IAP Callbacks

Product-specific events are defined in the **IAPListener** script and its 'HandleSuccessfulPurchase' method. If you want to add your own product events, simply add their global identifier (not the platform overrides) to the switch-case statement. This is the part where you say what happens when a user buys a product.

The **DBManager** manages our PlayerPrefs database and keeps track of purchases, selections, virtual currency and other player-related data using the JSON format in order save key-value pairs as a single string. Your app data is separated into three main categories: player-related data, content, selected products and currency.

The IAPListener has examples for increasing virtual currency and showing feedback messages:

```
DBManager.IncreaseFunds("coins", 1000); //increase coins by 1000
ShowMessage("1000 coins were added to your balance!"); //show feedback with text
```

In your game, you can check if a product is purchased by using:

```
DBManager.IsPurchased("your product id"); //returns a boolean
```

If you want to save your own app-specific data, DBManager provides methods for this purpose:

```
DBManager.SetPlayerData(string id, JSONData data);

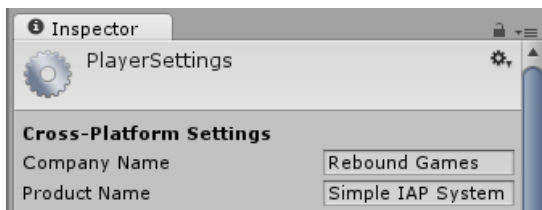
DBManager.SetPlayerData("score", new SimpleJSON.JSONData(2250)); //store user's score
DBManager.IncrementPlayerData("level", 1); //increase current level by 1
DBManager.GetPlayerData("level").AsInt; //get the level value as integer
```

Note that you need to put `using SIS;` (C#) or `import SIS;` (UnityScript) at the top of your scripts to use the Simple IAP System namespace. For a full list of available methods, please visit the scripting reference.

6 Encrypting device storage



Optionally you can encrypt the values created by DBManager on the device. On the IAPManager prefab inspector, enable “encrypt” to do so. Also, do not forget to replace “obfuscKey” with your own encryption key (8 characters on iOS/Android, 16 characters on Windows Phone 8). While other techniques are more secure, many App Stores require an encryption registration number (ERN) when submitting your app with those standards. This technique does not require an ERN. If you are about to submit your app to Apple's App Store and Apple asks you whether your app contains encryption, click YES. If they ask you whether your app qualifies for any exemptions, click YES again and you're done.

In development stages you can look up the stored data by opening your registry. The exact location depends on your Unity Project Settings:



Open up your registry, then find the corresponding app entry (in this case it's under Rebound Games > Simple IAP System).

Not encrypted vs encrypted data:

 data_h2087377941	{"Content":{"no_ads":"false", "abo_monthly":"false", "bonus":"false", "pistol":"true", ...
 data_h2087377941	DHdCnJfciE/vPhR5spmdWoYZVe/xiWi47LkMxkbtwnm1T1Ko37Siej7Ka9A9aO3++52k

It is good practice to clean up that entry between IAP testing. Press the “Clear Database” button in the AllSelection scene to delete all app-specific data set by Simple IAP System.

WARNING: Please be aware that our PlayerPrefs database implementation (DB Manager) may require a one-time only setup of variables. If you change their values again in production (live) versions, you will have to implement some kind of data takeover for existing users of your app on your own. Otherwise you will risk possible data loss, resulting in dissatisfied customers. Examples:

- Renaming or removing a virtual currency in the IAP Editor
- Renaming or removing IAPs in the IAP Editor
- Renaming internal storage paths in the DB Manager
- Toggling the encryption option in the DB Manager

Rebound Games will not be liable for any damages whatsoever resulting from loss of use, data, or profits, arising out of or in connection with the use of Simple IAP System.

7 Shop Templates explained

For a unique design, it is highly recommended to import your own images and build shop scenes and IAP Item prefabs to match your game's style. This improves conversion rates (free to paid).

- **List** (example scene 'Vertical'/'Horizontal'): these are the most basic samples in SIS and only contain products for real money, no in-game content or currencies. They have a 'Window – IAP' game object with ScrollRects ([?](#)), as well our IAPContainer component attached to the container. IAPContainer dynamically adjusts the cell size of its GridLayoutGroup according to aspect ratios and resolutions of different devices, as well as the width/height of IAP Item prefabs in the scene. IAPContainer also allows you to set the max width/height of an item in case of higher resolutions.
- **Tabs** (example scene 'VerticalTabs'/'HorizontalTabs'): when a single ScrollRect is not enough, maybe several are! These scenes also contain some in-game content products and currencies. Each category has a separate window in the scene, which get activated once the corresponding button on the left side is triggered. For example, the button for Items enables 'Window – Items' but disables all others. These scenes also display amounts of the 'Coins' currency at the top, via the 'UpdateFunds' script attached to a Text component.
- **Menu** (example scene 'VerticalMenu'/'HorizontalMenu'): these are the most complex scenes in SIS and showcase all product types. Instead of just enabling windows, here they are animated. Each window has an Animator component that moves it in or off screen. For example, clicking on the Items button will animate 'Window – Main' off screen and 'Window – Items' will show up. This behavior is handled per button, via two events set up on them. The 'Button – Back' in each window does the exact opposite. Also, pressing 'Button – LevelUp' in the Custom sub menu increases the user level and Shop Manager tries to unlock new items.

8 Receipt Verification

With your IAPs set up, you may want to add an extra layer of security to your app, which prevents hackers to just unlock items by using IAP crackers, sending fake purchases or simply overwrite its local database storage. Receipt verification could help at fighting IAP piracy. With Simple IAP System, you have two options on how to use receipt verification in your app.

Client-Side Receipt Verification

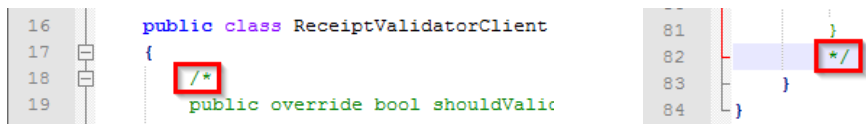
(Currently only supported when using Unity IAP)

This option utilizes the bundle and App Store developer key to verify that the receipt has been created by your app. While doing this check locally on the client's device results in a security flaw, this is a very fast approach to add some piracy prevention with just a few clicks:

1. Open Unity IAP's Obfuscator under Window > Unity IAP > Receipt Validation Obfuscator



2. Enter your Google Store Key (the same as entered on the IAPManager prefab).
3. Press on 'Generate', this creates additional obfuscated credential files in your project.
4. Remove the comments at the beginning and end of the file, so the script gets executed.



5. Locate the IAPManager prefab and add the ReceiptValidatorClient component to it.



Server-Side Receipt Verification

In this option, the receipt is sent to an external server, which forwards the transaction data to Apple or Google respectively, checks against it and then returns a valid or invalid response to the application. This means that the verification part only happens on the server. Also, if you want to check subscription status and restrict content for expired ones, you will have to use online receipt verification. In order to use this option, some configurations on your server are required, as well as in your iTunes/Google developer account. Due to the advanced settings and experience needed in setting this up, we are offering this as an additional service on our website. For more information, please find a link to our services page in the 'Contact' chapter.

9 Adjusting In Game Content remotely

Remotely hosted configuration files can be used to overwrite details of in-game content details, such as titles, descriptions or prices, without updating the app itself or to introduce temporary sales for products. The configuration file has to be defined in JSON format. See *SIS > Extensions > remote.txt* for a sample configuration. **Please note that if encryption is enabled, your hosted configuration will also be de-/encrypted.** After uploading the file to your server, enter the server url and file name to your configuration in the IAP Manager prefab. Supported remote types are:

- **Cached:** saves the new configuration on the device and loads it on the next app startup (permanent changes), which means that an internet connection is only required once
- **Overwrite:** only affects the current session (temporary changes), skipped on later startups

10 Localizing IAP data

If you've checked 'Fetch' in the IAP Settings editor, your IAP data will automatically be overwritten by the localized IAP details in the App Store. However, this only works for in-app purchases, not in-game content. Offline localization for all products is supported by integrating the free [Smart Localization](#) package. If you are unfamiliar with this package, please have a look at how it handles localization first. You'll find a short introduction [here](#), or a quick video [here](#).

- 1 Import the official [Smart Localization](#) package from the Asset Store.
- 2 Import our 'SmartLocalization' package found under *SIS > Extensions*.
- 3 If you haven't used Smart Localization before and would like to see a working configuration with examples, import the "SmartLocalizationWorkspace" package too.
- 4 Open the new 'Localization' scene in *SIS > Scene*. There's a special feature included: `LanguageSelection.cs` saves the active language between sessions in the storage.

Localize: In order to add products of Simple IAP System to Smart Localization's Root Language File, open the **IAP Localization editor** under *Window > Simple IAP System*. Select the items and fields you want to localize, then press the add/update button. Your products have now been added to the root file. Don't forget to save your changes in the Root Language window. You can now translate your product values in Smart Localization (*Window > Smart Localization*).

The last step is to attach the 'LocalizeIAPItem' component to your IAP Item prefabs. Expand its 'Fields' dropdown in the inspector and toggle the fields which should pull localized values. Now it is time to see your localization in-game! You can switch languages with the methods provided by Smart Localization and your shop items will update their values accordingly.

11 Contact

As full source code is provided and every line is well-documented, please feel free to take a look at the scripts and modify them to fit your needs.

If you have any questions, comments, suggestions or other concerns about our product, do not hesitate to contact us. You will find all important links in our 'About' window, located under *Window > Simple IAP System*.



If there are any questions, maybe our [FAQ and support forum](#) or [Unity thread](#) has the answer!

For optional, advanced integrations which will save you some time and further drive the conversion rates of your app, please have a look at the services offered on our website:

SERVICES

For private and/or open questions, you can also email us at
info@rebound-games.com

If you would like to support us on the Unity Asset Store, please write a short review there so other developers can form an opinion. Again, thanks for your support, and good luck with your apps!