

Simple IAP System (SIS)

by Rebound Games

v2.4.2

1. Scripting Reference	1
2. Introduction.....	2
3. First Steps.....	3
4. Setting up IAPs.....	5
3.1. Android.....	5
3.2. iOS	6
5. Scripts	9
4.1. IAP Manager.....	9
4.2. IAP Editor	10
4.3. Shop Manager	14
4.4. IAP Item	14
4.5. DB Manager	17
4.6. IAP Listener	19
6. Templates.....	20
5.1. List	21
5.2. Tabs	21
5.3. Menu	22
7. Online Receipt Verification.....	23
8. Remote Config Download.....	25
9. Product Localization.....	26
10. Contact	27

Thank you for your purchase!
Your support is greatly appreciated!

1. Scripting Reference

You can always find the latest scripting reference here:

<http://www.rebound-games.com/docs/sis/>

2. Introduction

Simple IAP System (SIS) takes the complexity out of in app purchases (IAPs) in your apps and delivers an easy-to-use framework for player monetization, packed with effective editor widgets, shop item generation at runtime, advanced database functionality and much more:

- Ready-to-use, multi-platform IAP wrapper for iOS and Android
- Option for overwriting item data with online data from Google or Apple
- Handles purchases with real money, but also virtual currency
- Comes with 6 shop templates built with uGUI/NGUI
- Online receipt verification on your external server, PHP scripts included
- Documentation on how to configure your Google/Apple developer account

There are 5 different IAP types in SIS:

- Consumables: can be purchased multiple times, for real money (e.g. coins)
- Non-consumables: one time purchase for real money (e.g. bonus content)
- Subscriptions: periodically bills the player, for real money (e.g. service-based content)
- Virtual consumables: same as consumables, but for virtual currency
- Virtual non-consumables: same non-consumables, but for virtual currency

As listed above, some IAPs require billing the player. Rather than reinventing the wheel, SIS makes use of 3rd party billing plugins for handling billing transactions. Virtual IAPs do not require 3rd party plugins, however you will need to use our database manager or a similar implementation for it to work.

SIS currently supports the following billing plugins:

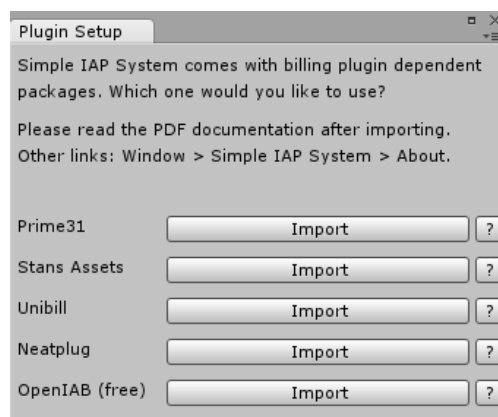
- [Prime31](#) (Google In App Billing, StoreKit (In App Purchasing), Combo)
- [Stan's Assets](#) (Android Native Plugin, iOS Native and Ultimate Mobile)
- [Unibill](#) (all Unibill-supported platforms)
- [OpenIAB](#) (Google Play, iOS App Store, Amazon (beta), Windows Phone 8 (beta))

3. First Steps

If you are new to Unity, please take a quick break and get dirty with its main functionalities first, because this documentation will assume you have some basic knowledge regarding the interface and its editor tools. You will find basic instructions on how to get up and running with this package below. For video tutorials, please visit our [YouTube channel](#).

🔑 Import Simple IAP System into your project (you have probably done this by now). You will see the contents of SIS, being scripts, example scenes and prefabs along with their textures.

🔑 When your project has finished compiling you will see the following editor window, which asks you to import your chosen billing plugin package. If it didn't show up, you'll find this window under Window > Simple IAP System > Plugin Setup. Import the desired package.



Plugin-specific instructions:

- Prime31: Import your Android and/or iOS billing plugin or the combo package.
- Stan's Assets: Import your Android/iOS Native plugins or Ultimate Mobile.
- Unibill: Import your Unibill package (1.3.10 or above). If you would like to use the IAP inventory that we used in this documentation, open and overwrite the default Unibill inventory with the contents of our file located in SIS > Plugins > unibillInventory.
- OpenIAB: Grab the latest release [here](#).

Deployment:



If you would like to run the example scene “AllSelection” later on, make sure to include all sample scenes in your Build Settings first.



Also, don’t forget to check your AndroidManifest.xml (Android) or StoreKit.framework (Xcode) before deploying to an actual device. Some of the supported billing plugins will do this for you (Prime31, Stan’s Assets), others won’t (Unibill, OpenIAB).

Let me introduce you to the basic structure and important components included in SIS:

- IAP Manager: cross-platform wrapper for initiating all the different types of IAPs
- IAP Editor: editor widget for managing IAPs and virtual currency
- Shop Manager: instantiates IAP Item prefabs and unlocks previous purchases/states
- IAP Item: represents an IAP item created in the IAP Editor as shop item at runtime
- DB Manager: saves purchases, selection states, virtual currency and player data
- IAP Listener: handles IAP callbacks and tells your app what to do for each product
- Templates: you can choose between 6 predefined shop GUI layouts. Lists, tabs or menus – each of them with a vertical and horizontal alignment

Preparing your scene for SIS is easy – just drag & drop the IAP Manager prefab found under *SIS > Prefabs > Resources* into the very first scene of your game, then drag & drop the Shop Manager prefab into your shop scene. Always keep the IAP Manager prefab in the Resources folder. When making changes to the IAP Settings later on, these changes are automatically saved to this prefab. Thus, if you are upgrading to a new version of SIS, make sure to keep a local copy of your IAP Manager prefab.

The next chapter shows how to create your app and IAPs in the Google/Apple developer center, before you can display them in your app. You can skip this chapter if you’re already up and running with IAPs.

4. Setting up IAPs

Please refer to the following chapters for registering you as a mobile developer, creating your first app and setting up IAPs in the respective developer center. This documentation only describes parts which are relevant for the in-app purchase process and does not explain how to build or distribute your app to a mobile device.

4.1. Android

Android Developer Program: \$25 registration fee.

<https://play.google.com/apps/publish/>

Google makes the registration process quite easy. Before entering the developer program, you will need a Google account. After signing in with your Google account and opening the registration page mentioned above, you will have to enter information about your identity, accept the agreement and create a Google Wallet account, if you don't have one. Google Wallet is used for paying the registration fee as well as receiving revenues from your apps. You can find more details on the registration process here:

<http://developer.android.com/distribute/googleplay/publish/register.html>

Once you've registered, log in and verify your email for testing access (under Settings). Then, navigate to "All Applications". Add a new application by selecting the language, provide a title and click on "Prepare Store Listing". On the next page, enter the necessary information about your app and save your entry. More details on that page here:

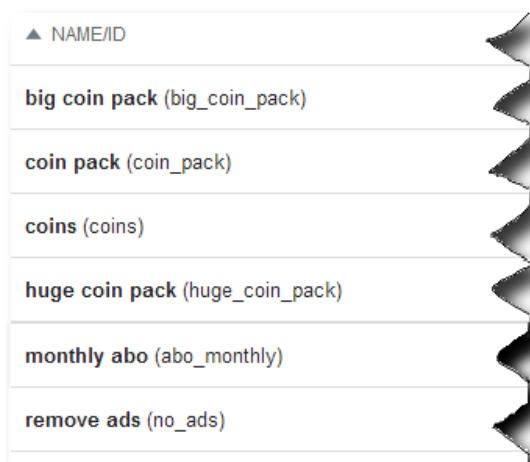
<http://developer.android.com/distribute/googleplay/publish/console.html>

Now, click on "In-app Products". It will tell you that your app doesn't have any in-app products. On this stage, Google needs an APK file that includes the permission "BILLING". Return to Unity, include all scenes from SIS and build the sample project to an APK. Again, the Unity APK build process is not explained here, but you will find various tutorials on the internet for that matter. Don't forget to change the app-related Android data in Unity before building, such as the bundle identifier found under Edit > Project Settings > Player, Settings for Android, Other Settings. Then, upload this APK to Google. You should now be able to create in-app products for the app.

To get the SIS sample project including its sample IAPs working, you will have to create the exact same in-app products in your Google Play developer console as present in SIS. Here is a link to the IAP Editor and its IAPs (discussed in one of the next chapters in detail, but referenced here for convenience): [Link](#). Please take a look at the first screenshot found there and create a new product in the Google Play developer console with the given information per product.

For example: Click on “Add new product”. Select “Managed product” for consumable and non-consumable items and “Subscription” for subscriptions. Enter the product id, in this case the first product id of SIS would be “coins”. Enter “coins” as the title and “1000 coins to unlock items.” in the description field, as shown in the referenced IAP Editor image. Lastly, enter “1,99” as price and save the product by activating it.

Note: If you can’t enter a price, you would first need to specify which countries and territories you want to distribute to. This setting is located under the “Pricing and Distribution” tab.



Repeat this process until you’ve created a separate in-app product for each of SIS’ IAPs (but not for the “restore” product) and your in-app product overview looks like this, with all of them being managed products and “monthly abo” being the only subscription-based product.

That’s it for the Google/Android part. One last important point is your license key (also called Google app key), which can be found under the “Services & APIs” tab. If you want to test the sample project on your device now, enter this key on each IAP Manager as described [here](#).

Important: With recent changes on Google Play, your app has to be published as Alpha or Beta build in order to test in app purchases! [Draft apps are no longer supported.](#)

4.2. iOS

iOS Developer Program: \$99 annual fee.

<https://developer.apple.com/programs/ios/>

Setting up an Apple developer account for iOS can be daunting, to say the least (it took Apple 2 months for our account from registration to activation). First of all, you'll need to create an Apple ID before you can enroll in the iOS Developer Program.

When applying for one of the developer programs, you have to choose an enrollment type (individual/single person or company/organization). Companies need a few more things, like the D&B D-U-N-S Number: <http://www.upik.de/en/start.html#>. This may take up to 30 days. Make sure that your legal entity name on Apple's side matches your D&B entry. Apple will then verify your information, send you a legal agreement and after that you are allowed to pay the applicable fee.

Similar to the Android part, this documentation does not describe how to deploy iOS apps in detail. Just let me tell you that you'll need Xcode installed on your Mac. Also, there are multiple steps involved before deploying to an actual device. You have to register your device as a test device in your Apple developer account, create certificates and provisioning profiles, what includes mapping your device to these profiles. Since there are various tutorials available on how to approach device provisioning, I'll point you to one of them that will fully guide you to the process. It's from the Xamarin docs, so please ignore possible statements about Xamarin on this page (follow steps 1-4): [Link](#).









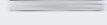



This section will guide you through the process for creating in-app products, but here is another resource from the Xamarin docs which explains the structure and workflow of Apple's in-app products pretty well (for general reference): [Link](#).

Log into iTunes Connect for managing apps, products, test users and more:

<https://itunesconnect.apple.com/>

Before creating in-app products, Apple wants you to enter banking information. Click on "Contracts, Tax and Banking" from the main menu and agree to the "iOS Paid Applications" contract on that page, then enter the information necessary until every section displays a green light. With the Explicit App ID created previously (described in both resources mentioned above), you can now click on "Manage your Applications" from the main menu. Select your app and click on "Manage In-App Purchases". Now we are able to create the sample products of SIS. Here is a link to the IAP Editor and its IAPs (discussed in one of the next chapters in detail, but referenced here for convenience): [Link](#).

Please take a look at the first screenshot found there and create a new product in iTunes Connect with the given information per product, until your app's IAP list look like this:

6 In-App Purchases Search				
Reference Name	Product ID	Type	Apple ID	Status
big coin pack	big_coin_pack	Consumable		 Waiting for Screenshot
coin pack	coin_pack	Consumable		 Waiting for Screenshot
coins	coins	Consumable		 Waiting for Screenshot
huge coin pack	huge_coin_pack	Consumable		 Waiting for Screenshot
monthly abo	abo_monthly	Non-Renewing		 Waiting for Screenshot
remove ads	no_ads	Non-Consumable		 Waiting for Screenshot

Please refer to the Xamarin docs pages linked above for further details and screenshots for each step. Lastly, because developers usually don't want to spend real money on test purchases, we have to add test users in iTunes Connect. Click on “Manage Users” from the main menu and add one test user or two. The name or email doesn't really matter, as they are just used for verification purposes. Don't forget to change the app-related iOS data in Unity before building, such as the bundle identifier found under Edit > Project Settings > Player, Settings for iOS, Other Settings. When testing IAPs on your device make sure to log out current users in the device's settings before starting your app. It will then ask for user details on the first purchase, so you can enter test user credentials. If the product shows up as “[Environment: Sandbox]”, you're good to go.

5. Scripts

This section will introduce every important component and its mechanics included in SIS.

WARNING: Please be aware that some of the following components, especially our PlayerPrefs database implementation (DB Manager), may require a one-time only setup of internal 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.

5.1. IAP Manager

Script Connections: none

You only have to enter your Google app key in the inspector (except Unibill), if you want to use Android as a publishing platform. That's all it takes at this point, receipt verification and remote config download will be discussed later. IAP Manager holds lists for every aspect of IAPs, such as product ids, product properties, virtual currency and the like. It initializes your billing plugins at game launch and fires IAP events when the initialization or a purchase failed and when a purchase succeeds. Normally you don't call methods of IAP Manager directly, unless you want to implement them on your own. We'll see which methods are in use when we investigate the IAP Item component in the following chapters.

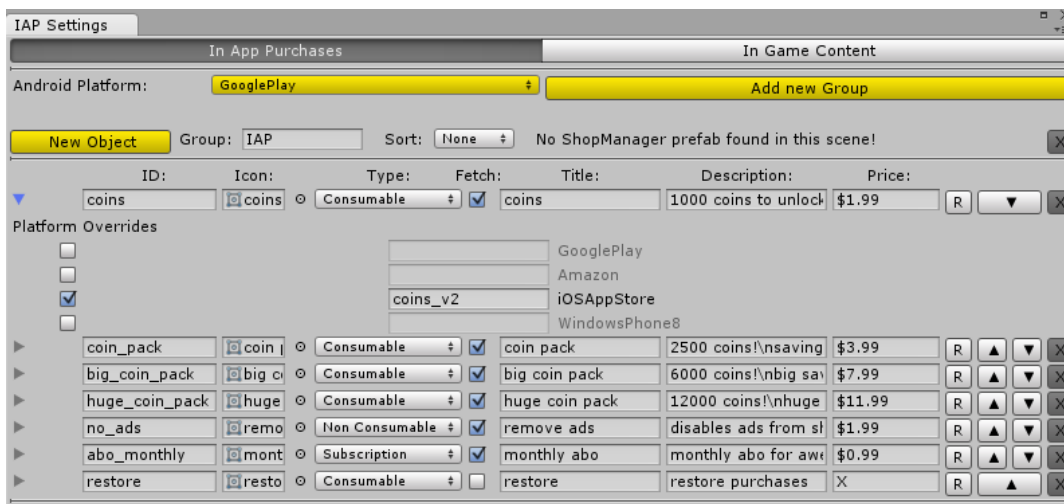
5.2. IAP Editor

Script Connections: IAP Manager

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



Please navigate to SIS > Scenes and open the example scene “Vertical”. Open up the IAP Editor, which is located under Window > Simple IAP System.



You can switch between real money and in-game items by clicking on the tabs at the top. When creating IAPs that charge real money, there's a foldout for overriding identifiers per store in case you have different product identifiers for the same IAP. **Unibill only:** Instead of foldouts, there is an additional button for importing items from Unibill. Items for real money must be defined in the Unibill Inventory Editor and then imported in the IAP Settings editor. Other items, such as virtual currency and virtual products, have to be defined in our window.

IAPs, more specifically IAPObjects, are grouped into IAPGroups. Each IAPGroup can hold an unlimited amount of IAPObjects. Furthermore, each group has a unique set of variables:

IAPGroup:

Group: Group name that must be unique across all groups (per platform). When using selectable items, the product id along with the group name are stored in the database for remembering selection states. For single selection, where only one selection per group can exist, this ensures all other items in the same group are being deselected. Likewise, if you only want to allow one selection across two groups, they have to share the same name.

IAPGroup:

Sort: Allows for sorting IAPObjects within the group. The following sort modes are supported: price ascending, price descending, title ascending, title descending.

Prefab: IAP item GUI prefab that holds the IAP Item component. This prefab gets instantiated by the Shop Manager per IAPObject and then filled with its information.

Container: Parent transform for all prefab instances in this group. In most cases this transform will hold a UITableView or UICollectionView component for automatic alignment.

▲ ▼: Re-orders the current group and moves it up or down, if there's more than one group.

X: Deletes the current group and removes all IAPObjects contained in it.

IAPObject: Represents an IAP product. If it's an IAP for real money, the item data should match to the product information entered in your Google/Apple developer center.

ID: Must be unique across all IAPObjects. Real money IAPs: your IAP product id from the Google/Apple developer center. Can be overwritten on a per-platform basis (open foldout).

Icon: Sprite reference for the icon of this item prefab.

Type: IAP type. The following types are supported: Real money IAPs: consumable, non-consumable, subscription. Virtual money IAPs: virtual consumable, virtual non-consumable.

Fetch: Whether local item data should be overwritten once online data from Google/Apple has been loaded. Overwritable fields are title, description and price (real money IAPs only).

Title: Product name to display.

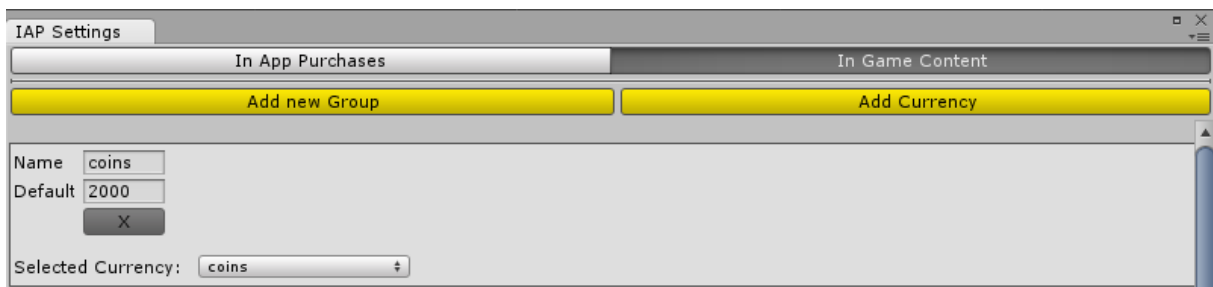
Description: Product descriptions support line breaks (insert "\n").

Price: Product price. Real money IAPs: string value. Virtual IAPs: int value. The item will start as purchased, if the price value is equal to 0 (virtual money IAPs only).

R: Opens the requirements window. Typically used for locked items. You can specify a requirement the user has to meet to unlock the item. See the IAP Item section for details.

Notice for “In Game Content”

Before you can modify prices for your virtual items on the “In Game Content” tab, you will need to specify one or more virtual currencies by pressing “Add Currency”. As with modifications to IAPGroups or IAPObjets, it is not recommended to rename or remove a virtual currency in production versions, as this could lead to inconsistency and loss in funds for users. A warning will appear if you try to do so.



Currency:

Name: Currency name. Must be unique across all currencies.

Default: Starting amount given to the player, when a currency is initialized for the first time.

Selected Currency: Currency selection field. You can enter IAP prices for the active selection.

The example scene “Vertical” showcases only a few IAPs for real money, but you can open the other scenes for more complicated setups and virtual IAPs. We will go through them in one of the next chapters. Just as a quick summary:

- You can manage your IAPs via the IAP Editor
- Each IAPGroup has a GUI prefab, which gets instantiated and then populated with local or online data per IAPObjets at runtime
- For selectable items, the IAPGroup name plays an important role for data storage
- IAPs for virtual money require at least one virtual currency
- After instantiating item prefabs, they are parented to an container transform and then auto-aligned in your shop layout via an Uitable or UGrid component

Steps for creating your own products in SIS

In chapter 3, we discussed how to set up your Google and/or Apple developer account in order to get the supplied IAPs of SIS working. Adding your own IAPs is equally easy. As a quick summary, this process can be broken up into 4 parts:

1. Specify IAPs in your Google/Apple developer account

Create your products online in the Google or Apple developer center respectively (skip this step for virtual products). Optionally create localized product information there.

2. Enter the exact same information in your IAP Editor

Open the IAP Editor and create a new object for your product. If there aren't any groups, you will need to add an IAP group first. When creating real money products, it is recommended to enter the same information for your product as presented online - by checking "Fetch" in the IAP Editor, your product information will be overwritten by online data (localized, if provided). Virtual products don't get overwritten in SIS, as they only exist offline.

Exception: "restore". For restoring real money purchases in your store (Google/Apple), 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 methods for you.

3. Optionally create your own IAP Items and choose a shop template

For a unique design, you will likely want to import your own images and build an IAP item as representation for your product based on these sprites (described on the following pages) and choose one of the existing shop templates or build your own.

4. Playtest

Most important, playtest your IAPs! Nothing damages your app more than dissatisfied customers. If you use our database DB Manager for saving previous purchases, you can reset purchases by deleting the associated registry entry or PlayerPrefs file to ensure a clean test state between testing stages (described later).

Please note that the IAP Manager and DB Manager will have to initialize themselves in-game before you can actually access any of their values (e.g. virtual currency). Thus, it is best practice to put them in your first game scene. Next, we will shortly discuss the Shop Manager component, which is responsible for instantiating item prefabs.

5.3. Shop Manager

Script Connections: IAP Manager, DB Manager

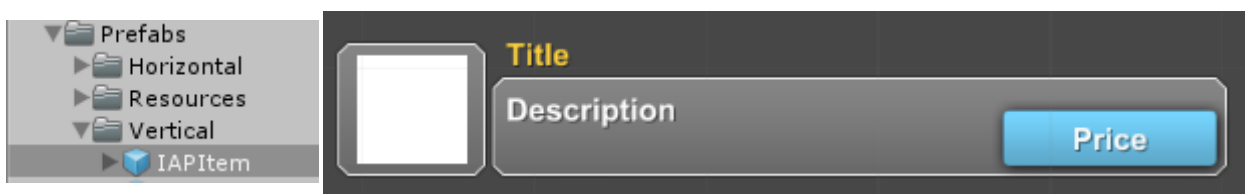
As mentioned in the last section, the Shop Manager component instantiates item prefabs, which visualize IAP products in your shop at runtime. It does that by wrapping IAP product information in a cross-platform class and then sets all the prefab GUI labels accordingly. After initializing all productions within the shop, the Shop Manager receives lists of purchased and selected products from the database. These lists are then used for displaying the appropriate item states in your shop to the user. This means that an already purchased item does not show the buy button, or a selected item has a button to deselect it and so on. Item-specific states are handled by the IAP Item component itself. As public variables, the Shop Manager exposes references to a GameObject and a label, which are being used for showing the feedback window along with a descriptive text in case purchases succeed or fail.

5.4. IAP Item

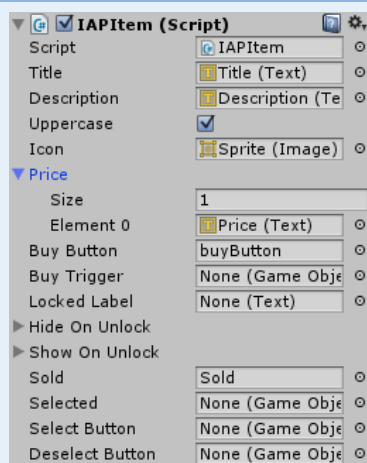
Script Connections: IAP Manager, Shop Manager

Item prefabs visualize IAP products in your shop GUI 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 Item should show or hide different portions of the prefab instance.

If you are using NGUI, the product title in the IAP Settings editor determines which texture gets chosen at runtime. Make sure the icon is in the correct atlas too ([see here for a video](#)).



IAP Item exposes the following variables:



Title: Label for the product title.

Description: Label for the product description.

Uppercase: When enabled, converts text to uppercase.

Icon: Image reference to the product icon.

Price: Label(s) for the product price. You can specify one for real money IAPs and multiple for virtual money IAPs.

Buy Button: Button GameObject for the buy button.

Buy Trigger: Additional Button GameObject for showing/hiding the buy button (optional).

Locked Label: Label that displays the requirement description for locked items.

Hide On Unlock/Show On Unlock: GameObject(s) to hide/show on item unlock.

Sold: GameObject that gets activated if this product has been purchased (purchased state).

Selected: GameObject that gets activated if this product has been selected (selected state).

Select Button: Button GameObject for selecting this product.

Deselect Button: Button GameObject for deselecting this product.

These are the IAP Item states supported:

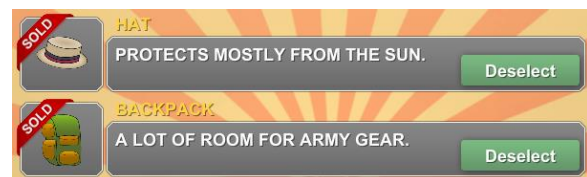
- Locked: For items with requirements (e.g. reach level 1 to unlock)
- Default: upon initialization or after unlock
- Purchased: buy button hidden, selection possible (excludes consumables)
- Selected: purchased and selected (excludes consumables)

Selection of non-consumable items is possible in three different ways:

- No selection: one-time purchase only, no further actions
- Single selection: selecting an item deselects all other items within the same group
- Multi selection: selecting an item shows the deselect button, multiple selections possible within the same group



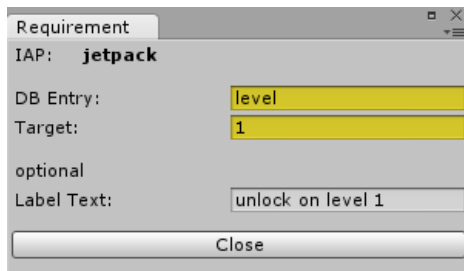
No selection (bonus level), single selection (weapons) and multi selection (items) in action



You can specify the selection type by assigning references to the IAP Item component. Every item prefab at least needs descriptive labels, such as a label for the title, description, etc. as well as a buy button. Other settings are up to you. So, for example, if you don't want the item to be selectable, don't assign buttons to the "selected" and "deselected" slot. If you would like to have a single selection interface, assign the "selected" button, but leave the "deselected" slot empty. For multiple selections, assign both "selected" and "deselected" button in the inspector. After setting up your prefab instance in the scene view, apply your changes to your prefab.



When setting up locked items, you have to prepare the prefab with visuals for the locked state, meaning you disable the buy button, description etc. to begin with, leaving nothing else than the locked visuals enabled (see the image above). For displaying the requirement description, assign a Text label to the "Locked Label" variable of IAP Item. If the item gets unlocked, it should hide all of the locked visuals and present the unlocked ones instead. Therefore you have to assign the gameobject with locked sprites to the "Hide On Unlock" variable and the unlocked ones to "Show On Unlock". You can find a sample use-case for locked items in the vertical/horizontal Menu scenes.



In these scenes, we set up a requirement for the jetpack item in the IAP Settings editor. “DB Entry” is the name of the DB Manager entry for the requirement. “Target” is the integer value the entry has to reach for unlocking this item. “Label Text” is the

text that goes in the “Locked Label” label of IAP Item. For example, here we unlock the jetpack if the player reached level 1. The next page explains how to set this data in your app.

5.5. DB Manager

Script Connections: IAP Manager

This component manages our PlayerPrefs database and keeps track of purchases, selections, virtual currency and other player-related data. It isn’t recommended to change DB Manager’s settings during development due to possible loss in data (see chapter 4), thus its variables are private and only modifiable in the code. You can modify where or how data should be saved on the device. The DB Manager uses the JSON format in order save key-value pairs as a single string. This way, only one PlayerPrefs entry is necessary for storing all app-related data. Your app data is separated into three main categories: player-related data, content, selected products and currency. These category names can be changed in code.

DB Manager (code):

Player: Mainly, here goes your own app-specific data. E.g. requirement values for locked items, amount of power-ups, travelled distance, or other things like that. Accessed via **SetPlayerData(string id, JSONData data)**.

For example, if you want to store the user’s score (e.g. 2250), call this:

```
DBManager.SetPlayerData("score", new SimpleJSON.JSONData(2250));
```

On the other hand, if you just want to increment the user’s current level by 1, call:

```
DBManager.IncrementPlayerData("level", 1);
```

All primitive data types are supported by SimpleJSON. If you need to get the level later, call:

```
DBManager.GetPlayerData("level").AsInt;
```


Content: A list of IAPs. Not purchased ones, purchased and selected along with the IAPGroup name go here. These are being modified by IAP events, you should not alter the contents directly. You can check purchases via `isPurchased(string id)`.

Selected: Selected products, along with their IAPGroup name. When a product gets selected, it will be added. If it gets deselected, it will be removed from its corresponding group.

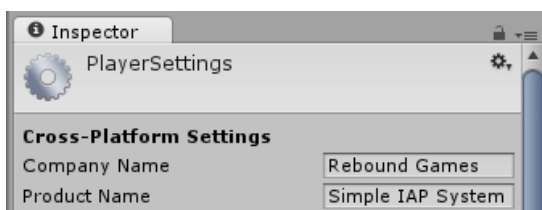
Currency: Currency names along with their current amount are getting stored here. Accessed via `GetFunds(string currency)` or modified via `IncreaseFunds(string currency, int value)`. Funds are being subtracted automatically when purchasing virtual products.

Optionally you can encrypt the PlayerPrefs string on the device. Please set “encrypt” to true 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 (because of the 64 bit symmetric key length) and you’re done.


Enabling encryption won’t stop hackers from reverse-engineering your code, finding out this key and decrypting the data to modify it, but it will prevent attempts to modify the data on jail-broken devices directly, because it’s not human readable anymore. Additionally, your users could copy-paste game saves from other users, but that’s not the main subject here.

 data_h2087377941 DHdCnJfcIE/vPhR5spmdWoYZVe/xiWi47LkMxkbtwnm1T1Ko37Siej7Ka9A9aO3++52k

In development stages and when encryption is disabled, 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 navigate to `HKEY_CURRENT_USER > Software` and find the corresponding app entry (in this case it’s under Rebound Games > Simple IAP System).

 data_h2087377941

{"Content":{"no_ads":"false", "abo_monthly":"false", "bonus":"false", "pistol":"true", ...

It is good practice to clean up that entry between IAP testing stages or to set a product state directly, if you want to test something. Other important variable definitions below:

DB Manager:

keepLegacy: (code) Whether or not old IAP entries on the user's device should be removed if they don't exist in the IAP Settings editor anymore. true: keeps them, false: removes obsolete purchases. Your users won't be too happy about an obsolete purchase though.

5.6. IAP Listener

Script Connections: IAP Manager, Shop Manager, DB Manager

The IAP Listener script is possibly the most important part of the IAP process, as it describes the final output of a purchase. It has references to each managing script to tell them what should happen on specific IAP events. Error messages are fired automatically through IAP Manager's purchase failed event and then forwarded to Shop Manager's error window.



Please open up IAPListener.cs in your favorite code editor

In HandleSuccessfulPurchase(), you tell your game what to do on a successful purchase. This method blocks further purchases of non-consumable products by setting their state to "purchased" and differs between product ids in a switch-case manner, so you can add your own product ids easily. For example, if the user has purchased some coins, you can increase funds and show the feedback window by using

```
case "coins":
    //the user bought the item "coins",
    //increase coins by 1000 and show appropriate feedback
    DBManager.IncreaseFunds("coins", 1000);
    ShowMessage("1000 coins were added to your balance!");
    break;
```

For more examples, please investigate the script directly. If you want to add your own products, simply replace this switch-case statement with your global product ids (not the platform overrides) and call the methods you desire:

```
switch (id)
{
    case "your product id":
        //what to do?
        break;
}
```


6. Templates

There are 6 shop templates included in SIS, divided into lists, tabs and animated menus. Each category comes with vertical and horizontal designs. The following chapters describe how these are set up and hopefully ease the work when configuring own designs.

Every template has an IAP Manager in the scene, populated with various IAP products, as well as an IAP Listener script to react on these events. Likewise, masked canvas are used for displaying and scrolling over groups of products and regular canvas draw the rest of widgets.

We will investigate the shop templates in SIS on the following pages.


6.1. List

 Please open the example scene “Vertical” or “Horizontal”.

These scenes are considered to be the easiest samples of SIS. If you open up IAP Settings, you will see just one group of IAPs. Only products for real money, no in-game content or virtual currency.

This scene contains one window for this group of IAPs, named “Window – IAP”. This window contains the Scrollbar and ScrollView panels, with a UIPanelStretch component attached to the Container gameobject. UIPanelStretch dynamically adjusts the cell size of its GridLayoutGroup according to aspect ratios and resolutions of different devices, as well as the width/height of IAPItem prefabs in the scene. UIPanelStretch also allows you to set the max width/height of an IAPItem in case of higher resolutions. A vertical/horizontal scroll bar is used if not all products fit to the screen.

6.2. Tabs

 Please open the example scene “VerticalTabs” or “HorizontalTabs”.

When using multiple IAP groups with different items, most likely a single scroll view for all of them won’t be enough. If you open up IAP Settings, you will see three groups in total, spread over real money IAPs and virtual IAPs. For virtual items, we have added a virtual currency named “coins” too. This scene features consumable, non-cons. and single selection IAPs.

Now, each group has a separate window in the scene (Window – IAP, Window – Items, Window – Armory). Also, there are three buttons for switching between windows. This happens by enabling the one selected and disabling the others. For example, if you want to switch from “IAP” to “Items”, the “Items” button disables “IAP”, “Armory” and enables “Items” via three events on the button itself. Please investigate the buttons to see how they are set up. Additionally, this scene displays the amount of “coins” on the top of the screen. Via the “UpdateFunds” script, you set a label for displaying the amount and type in the currency name, that’s it. The amount gets updated on every purchase (optionally animated).

6.3. Menu



Please open the example scene “VerticalMenu” or “HorizontalMenu”.

As seen in the Tabs example scenes, Menu scenes consist of multiple windows with multiple buttons and a virtual currency amount display. IAP Settings now holds 5 groups, one of them being for real money and four of them in the in-game content section. These scenes feature consumable, non-consumable, single selection, multi selection IAPs and locked items (that’s all there is) and are the most advanced scenes in SIS.

At a first glance, you will see the intro menu with 4 buttons, linked to each window. Instead of enabling or disabling windows like in the Tabs example, here they animate two windows at the same time to a specific direction. Each window has an Animator component for that purpose. For example, when clicking on “Items”, the menu window “Main” will animate off screen and “Items” will show up. This behavior is handled per button, via two events set up on them. The “back” button in each window does the exact opposite. Again, please investigate the buttons directly for further details.

Also, in the IAP Settings editor there are two items with requirements set up. If you press the “Level Up” button in the ‘Custom’ sub menu, the Shop Manager will try to unlock these.

7. Online 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 and sending fake purchases. This feature sends the receipt to your external server, which forwards the transaction data to Apple or Google respectively and then returns a valid or invalid response to the application. **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, you will have to configure some settings on your server, as well as in your iTunes/Google Developer account.

OpenIAB only: Receipt verification is currently not available on iOS due to missing receipts.

First, open the PHP script found under SIS > Scripts in the project panel. As values for “ios_bid” and “android_bid”, fill in your corresponding application bundle identifier from Apple/Google in the form of com.yourcompany.appname.

iOS: Look for the “\$secret” variable in the “verifyReceiptIOS” method. If you want to use and verify subscriptions in your game, you will have to set this key. Log into our iTunes Connect account (<https://itunesconnect.apple.com/>) to create one. Navigate to “Manage your Apps”, select your existing app entry, click on “Manage In-App Purchases” and click on “View or generate a shared secret” at the bottom of this page. Generate a new shared secret, then copy-paste this value in the PHP script.

Android: In order to use receipt verification with Android (Google Play Android Developer API), Google requires you to set up an authorized application. For more information, please refer to this link: <https://developers.google.com/android-publisher/authorization>. Log into Google Cloud as stated there. In the new Google Cloud Console, create a new project. After creating it, select it and navigate to “APIs & auth” > “APIs” and enable the “Google Play Android Developer API”. Go to “Registered apps” on the left side, then register a new web application. Now back to our PHP script, scroll down to the requestAccessTokenAndroid method and fill in your \$client_id and \$client_secret with the values from your newly registered application in Google Cloud.

Google uses an access token for verifying receipts that lasts one hour until expiration. Our PHP script will automatically request new access tokens before they get expired, but you will need a refresh token to request these. Still within your web application on Google Cloud, fill in your server URL and a callback parameter as value for “redirect URI”, for example “https://www.your-domain-name.com/oauth2callback”. Press “Generate”.

After this step, open the following link and replace it with the values you’ve received earlier.

```
https://accounts.google.com/o/oauth2/auth?response_type=code&scope=https://www.googleapis.com/auth/androidpublisher&redirect_uri=YOUR_REDIRECT_URI&access_type=offline&client_id=YOUR_CLIENT_ID
```

This site will ask you for permission to use the Google Cloud services, click “Allow access” when prompted. The browser will redirect you to your redirect URI and potentially throw a 404 error, but the important thing to notice is the code parameter at the end of the address. Important: save this key for the next step, where we actually request the first access token and refresh token. Open up www.hurl.it for creating a POST request and enter the following:

Destination
Follow redirects: [Off](#)

POST

https://accounts.google.com/o/oauth2/token

Authentication

+ Add Authentication

Headers

+ Add Header(s)

Parameters
[remove all](#)

grant_type	authorization_code	x
code	YOUR URL CALLBACK CODE FROM THE LAST STEP	x
client_id	YOUR CLIENT ID SEE GOOGLE CLOUD CONSOLE	x
client_secret	YOUR CLIENT SECRET SEE GOOGLE CLOUD CONSOLE	x
redirect_uri	YOUR REDIRECT URI SEE GOOGLE CLOUD CONSOLE	x

Add another parameter

⚡ Launch Request

Important: this step will only work once. After pressing “Launch Request”, scroll down and you will see a response that contains your refresh token. Now, copy its value and paste it as value for the \$refresh variable in our PHP script. You’re done!

Finally, upload the modified PHP script to your external server. It will create an additional file named “access_token.php” to save valid access tokens, make sure that your server grants write access in this directory. Back in Unity, select the IAP Manager and point the server URL and verification file name to your PHP script. Choose the verification type “onStart” to verify old purchases on game launch, “onPurchase” to verify new purchases, or both. Play-test your server verification process on the actual device with logging enabled.

8. Remote Config Download





Remotely hosted configuration files can be used to overwrite details of virtual products, such as titles, descriptions or prices, without updating the app itself or to introduce temporary sales for items. The configuration file has to be defined in JSON format. See SIS > Plugins > remote.txt for a sample configuration. After uploading the configuration file to your server, enter the serverUrl and remote file name path in the IAP Manager inspector.

You have the choice between two different remote types: cached and overwrite. **Cached** will save the new config file 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 and immediately applies the config file (temporary changes), but it does not get saved on the device and will be skipped on app startups without internet. **Please note that if encryption is enabled, your hosted config will also be de-/encrypted.**

9. Product Localization

If you enable “Fetch” in the IAP Settings editor, your IAP data will automatically be overwritten by the localized IAP information you’ve entered in the App Store. However, this only works for in-app purchases, not in-game content, and the user has to be online. Offline localization for all products is supported by integrating the free [Smart Localization](#) package.

Install instructions (in a new project with Simple IAP System):

-  Import the official [Smart Localization](#) package from the Asset Store.
-  Import our “SmartLocalization” compatible package found under SIS > Plugins.
-  If you haven’t used Smart Localization before and would like to see a pre-defined configuration with examples, import the “SmartLocalizationWorkspace” package too.
-  Open the new “Localization” scene in SIS > Scenes to see a working example. Feature: LanguageSelection.cs stores the active language between sessions in the DBManager.

Localize your own products:

First, you should get familiar on how Smart Localization handles localization. They have a short introduction to it [here](#), as well as a quick video tutorial [here](#). For more information, please see the official Asset Store page.

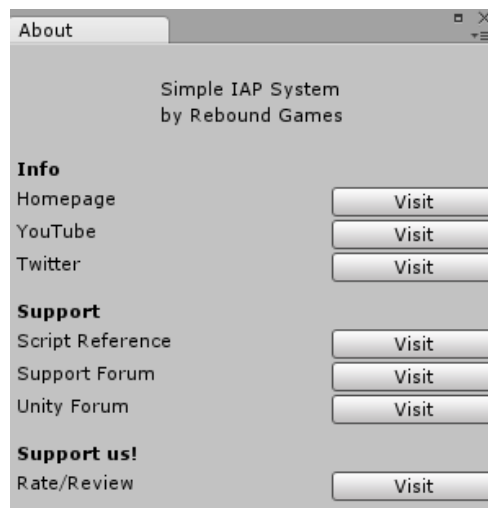
In order to add products of Simple IAP System to Smart Localization’s Root Language File, open the IAP Localization editor under Window > Simple IAP System. Select the items and fields you want to localize, then press the add/update button. Your products have now been added to the root file. Don’t forget 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 IAPItem prefabs. After you’ve done that, expand the “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 products will update their values accordingly. Hint: You can use the LocalizeUILabel component on regular UI Text components.

10. Contact

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

If you have any questions, comments, suggestions or find errors in this documentation, do not hesitate to contact us. You'll find all the important links in the "About" window, located under Window > Simple IAP System.



If you want to support us and in case you've bought this asset on the Unity Asset Store, please write a short review there so other developers can form an opinion!

If you have any private questions, you can also email us at

info@rebound-games.com

Again, thanks for your support,
and much success with your apps!

Rebound Games