Simple IAP System for SOOMLA Documentation

v1.1.4

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

Scripting Reference

www.rebound-games.com/docs/sisoom

1 Getting Started

Simple IAP System for SOOMLA (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 and customizable shop templates.

The use of SOOMLA products and features of SIS require an account on the SOOMLA platform.

1. Register for SOOMLA using this REGISTRATION LINK **(important)**
   SOOMLA Privacy Policy: (link)

2. Download the GrowSpend SOOMLA package (minimum setup): Download
   If you would like to use more functionality of GROW, find the other Unity bundles here.

3. Import the package you just downloaded into your Unity project.

4. On the GROW dashboard, add a new game and copy the game keys for your app. In Unity, open Window > Soomla > Edit Settings and paste them there. Also change the Soomla Secret and Google Play API key (if applicable).

5. Drag the "IAP Manager" prefab from *SIS* > *Prefabs* > *Resources* into the very first scene of your game.

6. 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).

DO NOT initialize SOOMLA's Store/Highway plugin in code, Simple IAP System does this for you! You're all set to create in-app purchases for your app! Follow these two steps:

- Create IAPs for your platform on the App Stores: e.g. Google Play, Amazon or iOS.
- Create your IAPs for real or virtual currency in Unity – see the next chapter!

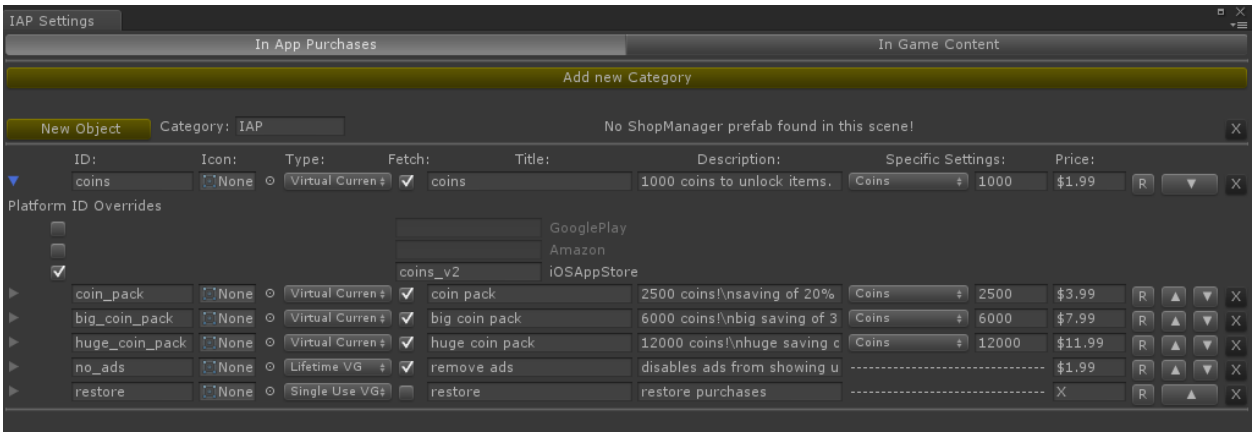
For a **video tutorial**, please visit our YouTube channel.

For a **sample integration and demo game**, please check out the scene located in SIS > Demo. To set it up, add the scene to the build settings. Deploy it to your device for testing!

2 Creating In-App Products 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.



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. **In Game Content**: here you create IAPs for virtual currency, depending on your game logic.

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

- **Category:** unique name of the group you are adding products to
- **ID**: 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 SOOMLA economy model by SOOMLA. If you are unsure which IAP type you need, please refer to the store model in SOOMLA's knowledge base.
  - VirtualCurrencyPack (buy coins, diamonds, or diamonds for coins)
  - SingleUseVG (consumable goods such as power-ups, etc.)
  - SingleUsePackVG (packs of a specific good, e.g. 5 power-ups)
  - LifetimeVG (non-consumable goods, e.g. remove ads)
  - EquippableVG (for example weapons or characters)
  - UpgradeVG (goods with multiple purchase levels, e.g. strength)
- **Fetch**: whether local product data should be overwritten by fetched App Store data
- **Title**, **Description**: descriptive text for the product in your shop
- **Specific Settings**: depend on the selected product type, as implemented by SOOMLA:
  - VirtualCurrencyPack: amount of currency that the user receives
  - SingleUsePack: amount of another good that the user receives
  - EquippableVG: category the equipping takes effect in, see store model for details
  - UpgradeVG: original good id, previous and next upgrade identifiers (empty if none)

- **Price**: product price (string value for real money, int value for virtual currency). Products in the 'In Game Content' tab 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 SingleUseVG and keep 'Fetch' unchecked. The IAP Manager will recognize this id and call the appropriate restore methods for you.

For products in the 'In Game Content' tab, 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.

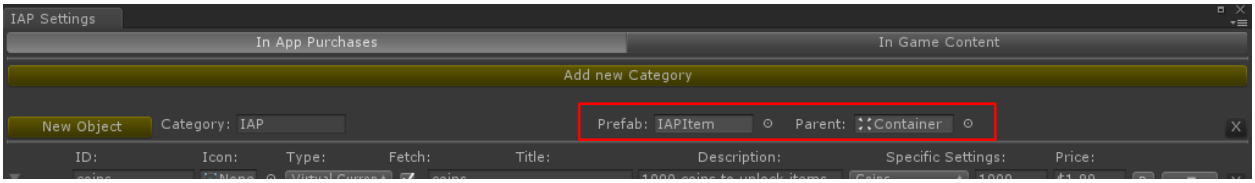
Side note: always keep the IAP Manager prefab in the Resources folder. When making changes to the IAP Settings editor, these changes are automatically saved to the prefab. Thus, if you are upgrading to a new version of SIS, make sure to keep a local copy of your IAP Manager prefab.

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 category in the IAP Settings editor, you only have to define what visual representation (**prefab**) and where (**parent**) you want to instantiate your products. There are several pre-defined 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.



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.

What happens with a purchase is defined in the **IAPListener** and its **HandleSuccessfulPurchase** method. There you can add your own messages for product identifiers in a switch-case manner:

```
case "coin_pack":
    ShopManager.ShowMessage("2500 coins were added to your balance!");
    break;
```

Virtual currency, as specified in the IAP Settings editor, is being added and substracted automatically for all products. For some other products, you need to actually define or query the result of successful purchases in your game logic. For the "no_ads" product, for example, you would have to check the item balance later in your game before showing ads, like this:

```
if(StoreInventory.GetItemBalance("no_ads") <= 0)
{
    //show ads
}
```

Other types, such as consumable goods or packs, would require checking the balance on certain user actions and reducing it in-game. For the "bullets" product, you could query the current balance and reduce it on every shot by calling `StoreInventory.TakeItem("bullets", 1);`

Note that when using products of type "Single Use VG", their amount only increases by 1 per purchase. If you would like to sell packs, e.g. 20 bullets, use a "Single Use Pack VG" instead.

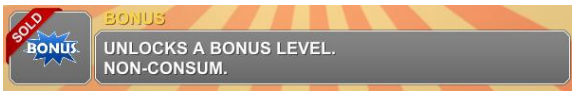
Finally, the IAPListener script has a method named **HandleFirstStart**, which gets called only once on the first launch of the app (in a lifetime). You can use this method to pre-define already equipped products before the player can select any of them – see the method for an example.

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)

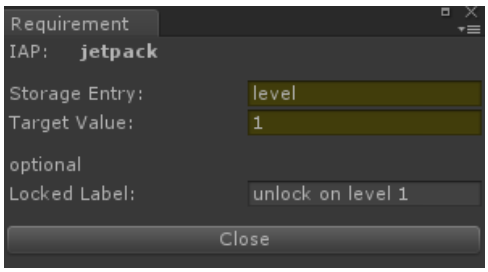
Equip: Category/Global (unequips others)

Equip: Local (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
- **Equip**: **Category/Global:** assign the 'Select Button', but leave the 'Deselect Button' empty
- **Equip**: **Local:** 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.

For setting the target value of a requirement, Simple IAP System makes use of SOOMLA's storage functionality. See the LevelUp() method in DebugCalls.cs for an actual implementation used for unlocking the jetpack product. E.g. if you want to unlock an item based on the player's current score, you would do this by setting the requirement's storage entry to 'score' and then modify it:

```
using Soomla;

public class TestClass : MonoBehaviour
{
    void SetScore()
    {
        int myScore = 1234; //player score
        KeyValueStorage.SetValue("score", myScore.ToString());
    }
}
```

5 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 UIPanelStretch component attached to the container. 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 IAP Item prefabs in the scene. UIPanelStretch 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.

6 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* > *Plugins* > *remote.txt* for a sample configuration. 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

7 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* > *Plugins*.
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 your products:

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.

8 FAQ

Q: The debug log on the device says that billing is not supported.

A: One reason could be that your bundle identifier in Unity does not match with the one on the App Store. When building for Google Play, your developer key must be entered in SOOMLA's settings and your app has to be published as alpha/beta build, in addition to being a test user.

Q: I'm getting some random 'This version of the application is not configured for billing' error.

A: Related to Google Play, there are multiple requirements before testing IAPs, see here.

Q: I want to reset my purchases, so I can start testing the same products again.

A: You can use the 'Clear Database' button (or its code) from the 'AllSelection' scene to clear all entries. For Google Play, you have to cancel the purchase in Google Wallet as described here and wait for a refresh or clear the cache of your Google Play app on the device manually.

Q: My shop items are duplicated or do not show up at all.

A: Make sure that the IAP Manager prefab is only placed in the first scene, and your prefabs and parents are assigned in the IAP Settings editor. Optionally see if there are errors on the device.

Q: The IAP Manager can't find my remote configuration file for overwriting in-game content.

A: If you've verified that it actually is on your server, check that the 'Server Url' field on the IAP Manager follows the pattern http://www.mysite.com/, with the file name being e.g. remote.txt.

Q: How do I customize and modify visual elements of an IAP Item prefab?

A: These prefabs consist of various UI elements introduced in Unity 4.6. You can find manuals and tutorials on the official site. Drag the prefab in the scene, make your changes and apply them.

Q: I want to have the same size for shop items on all devices, regardless of the screen resolution.

A: UI elements in SIS are using pixel-perfect images for best readability. You are looking for Unity's Canvas Scaler component (Scale With Screen Size). If you're using one of the sample shop scenes, you should also set both 'Max Cell Size X/Y' on the 'UIPanelStretch' components to 0.

Not answered yet? Maybe SOOMLA's FAQ has the answer!
If not, find our contact channels on the next page.

9 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*.

For private 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!

Rebound Games