MOBILE IAP

View online: https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/mobile-iap

The **Mobile IAP** plugin allows a game to use consumable and non-consumable **in app purchases** (IAP) on Android and iOS.

> *IAP can only be used in Android or iOS apps. IAP will not work in local preview, remote preview, or with any export option other than Android or iOS.*

Consumable vs Non-consumable

Non-consumable purchases can be bought once per user. A good use for this type of purchase is to distribute your game for free but with the majority of the content locked from the user. To unlock the rest of the content the user can purchase a premium upgrade. The plugin keeps track of these types of purchases for you across multiple user devices.

Consumable purchases can be bought multiple times by each user. A good use for this type of purchase is to enable a timed bonus or virtual currency. You need to keep track of these types of purchase yourself.

Setup

Before adding purchases to you game you will need to set up your application on Google Play and/or the App Store.

To register an app on Google Play visit https://play.google.com/apps/publish. To enable in app purchases you will also need to setup a Google payments merchants account at http://www.google.com/wallet/merchants.html.

To register an app on the App Store visit https://itunesconnect.apple.com/login

Registration stage

When making IAP events, the first thing you need to do is to complete the plugin's registration stage. You won't be able to make any purchases or check if a user owns a product until this is done. For each product you have you must call the *Add Product ID* action with the product ID and type (consumable or non-consumable). To finish registration you must call the *Complete Product Registration* action. If registration succeeds then *On Registration Success* triggers. After this you are free to check the state of products and make purchases.

Owned products may appear as "available" instead of "owned" in the "On Registration Success" trigger, but will resolve soon afterwards. So it's worth loading store logic before you need it. You can use the "On product owned" trigger to observe when a product becomes owned.

You only need to register the products once per session. Using the *Add product ID* or *Complete product registration* actions after calling registration will have no effect.

Displaying products

When offering a product to a user it is important to first ensure that it is available for purchase, otherwise the user will not be able to purchase it. A product may not be available if it is already owned, is invalid, has been flagged as unavailable by the store or (if consumable) is in the process of being purchased. This plugin also includes expression to get the name and description of a product from the store. It's important that your application uses these where appropriate rather than text that you have included in your application, as the app store is known to reject applications that don't follow this rule.

Keeping track of purchases

It's important to consider how you're going to keep track of user purchases before you start adding IAP events. Non-consumable purchases are tracked by the app stores, so you can easily find out if the user owns it. However, consumable products are a bit more tricky. The app stores don't track these, so you need to store information about these yourself. Initially it may seem appealing to store this information in Local Storage or similar, and while this works quite well for local testing it has a big issue: synchronization. If a user shares their account across multiple devices only the device that performed the purchase will know of it. This is also a pretty big issue if they move to a new device or reinstall your game, as they will lose content **they have paid for** and this tends to upset users. So best practice is to store this information in a remote database somewhere, that you can connect to from your game to check the purchases a user has made.

Exactly how you set this up very much depends on the game you are making and if you are offering a short timed bonus you may not even need a database. A simple setup would be a single number representing a user's balance of "magic gems" or similar. With a more complicated game you may choose to store the entire game state on the server; the world, balance of virtual currency, purchases of virtual goods made with virtual currency and purchases of virtual currency with real currency. A complete setup like this has the bonus of a user not losing virtual goods when moving device, and being able to play the game on multiple devices, but does increase your network dependence.

Testing

Both the app store and the play store offer mechanisms for testing purchases without spending real money. You should use these to confirm that your products have been correctly configured,

and that your purchase flow works correctly for successful and unsuccessful purchases as well as checking ownership on subsequent runs of your application. While the plugin unifies purchases for iOS and Android there are subtle differences in timings and product behaviour that you may experience. These timing differences can upset your purchase flow, so it's important to check any timing assumptions you make for one platform hold for the other.

Trigger order

Here are the sequence that triggers fire in for various scenarios.

Product not owned

On startup:

1. On registration success
2. On product available

On performing a successful purchase:

1. On purchase success
2. On transaction finished
3. On product owned

On performing a failed purchase:

1. On purchase failed

Product already owned

On startup:

1. On registration success
2. On product owned

IAP properties

Validator URL

Optional URL of a receipt validation service to verify in-app purchases with. This is for advanced users to implement a server to ensure purchases are valid. The URL is passed to the underlying `cordova-plugin-purchase`; refer to the Cordova plugin documentation on store.validator for technical details on how to set up or implement a validation service.

IAP conditions

On Purchase Success

Triggers when a specific product purchase succeeds.

On Any Purchase Success

Triggers when any product purchase succeeds.

On Purchase Failed

Triggers when a specific product purchase fails.

On Any Purchase Failed

Triggers when a any product purchase fails.

On Registration Success

Triggers when registration has been completed (after the *Complete registration* action). This is a good time to check if a product is owned. You should wait for this trigger before attempting any purchases.

On Registration Failure

Triggers when registration failed. If this occurs then you won't be able to make any purchases.

On Product Available

Triggers when a specific product becomes available to purchase.

On Any Product Available

Triggers when any product becomes available to purchase.

On Product Owned

Triggers when a specific product becomes owned. This triggers both after a first purchase for it, and on startup when the product was previously purchased, allowing easily identifying if the purchased product can be made use of.

On Any Product Owned

Triggers when any product becomes owned.

Product Owned

True if the current user owns the product.

Product Available

True if the current user can purchase the product.

Store Registered

True if the registration stage successfully completed.

On transaction finished

Triggered when the store transaction for a purchase has finished. The transaction ID is available in the *TransactionID* expression.

IAP Actions

Add product ID

Add a new product to the plugin by specifying the ID and type (consumable or non-consumable). This can only be called in the registration stage.

Complete product Registration

Ends the registration stage. After this has been called you will no longer be able to register products. This must be called before you can purchase products. *On registration success* will trigger if successful.

Restore Purchases

Restores user purchases. This is not necessary on Android.

Purchase Product

Triggers the purchase of a product with a specific ID. This product must be available to purchase. *On Purchase Success/Failed* will trigger depending on the outcome of the purchase. If the purchase is successful, *On product owned* will also trigger, as well as on startup in future sessions while the product is still owned.

IAP Expressions

ProductName(ProductID)

Get the name of a product from its ID. This is the localized name provided by the store, you should use it instead of a hard-coded string.

ProductPrice(ProductID)

Get the price of a product from its ID. This is the localized value provided by the store, you should use it instead of a hard-coded value.

ProductDescription(ProductID)

The description of a product from its ID. This is the localized string provided by the store, you should use it instead of a hard-coded string.

ProductID

The ID of the current product in a trigger.

TransactionID

In *On transaction finished*, the ID of the transaction that finished, using the ID provided by the store.

ErrorMessage

In an error trigger, the relevant error message, if any.