

## Fusion: Android In-App

**Application:** Clickteam Fusion (2.0 / 2.5)

**Applicable Exporters:** *Android*

**Time Required:** *~30 Minutes*

---

# AN EXTENSIVE GUIDE TO IMPLEMENTING IN-APP PURCHASES INTO FUSION APPS

Welcome to this walkthru tutorial on how to implement Android In-App Purchases into your Fusion applications and games. This walkthru was written on 27<sup>th</sup> November 2013 and could be subject to changes dependent on any changes with Google's IAP system.

You will require Multimedia Fusion 2 with the Android Exporter or Fusion 2.5 with the Android exporter. This walkthru should take no longer than an estimated 30 minutes to read through. Your final implementation of this example should also be around 20 minutes.

---

## CONTENTS

- 1) Setting Up Your App in Fusion
  - 2) Setting Up Your App in Google Play
  - 3) Design
  - 4) Implementation
  - 5) Testing
  - 6) FAQ (Frequently Asked Questions)
- 

This guide was written with everybody in mind. From entry level users of Fusion products through to advanced users. You can always check out the forums over at <http://community.clickteam.com> for further information, questions, answers and examples if you are looking for something that is not contained inside this guide. You can also refer to Google's Official Documentation on In-App Purchases as that is exclusively applicable to the IAP object you will be using.

As usual, all comments, suggestions and feedback is welcome on this guide.

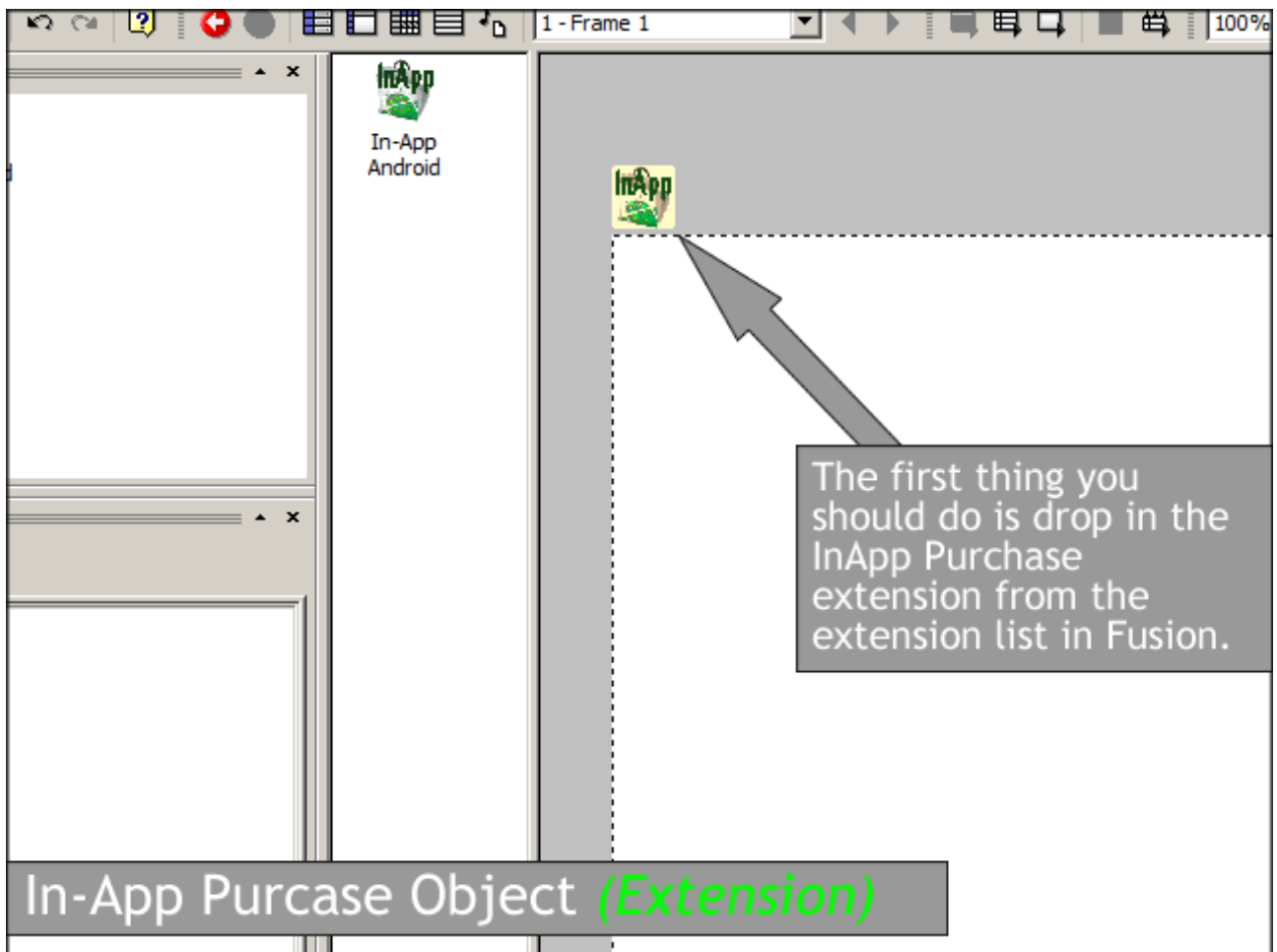
I hope you find this guide useful.

Enjoy!

Danny

## Fusion: Setup App in Fusion

The first thing you should do here is, setup your application inside Fusion. Any Android application that is going to be using In-App Purchasing requires the correct object and the correct permissions to enable the object to function correctly, so let's go ahead and do that.

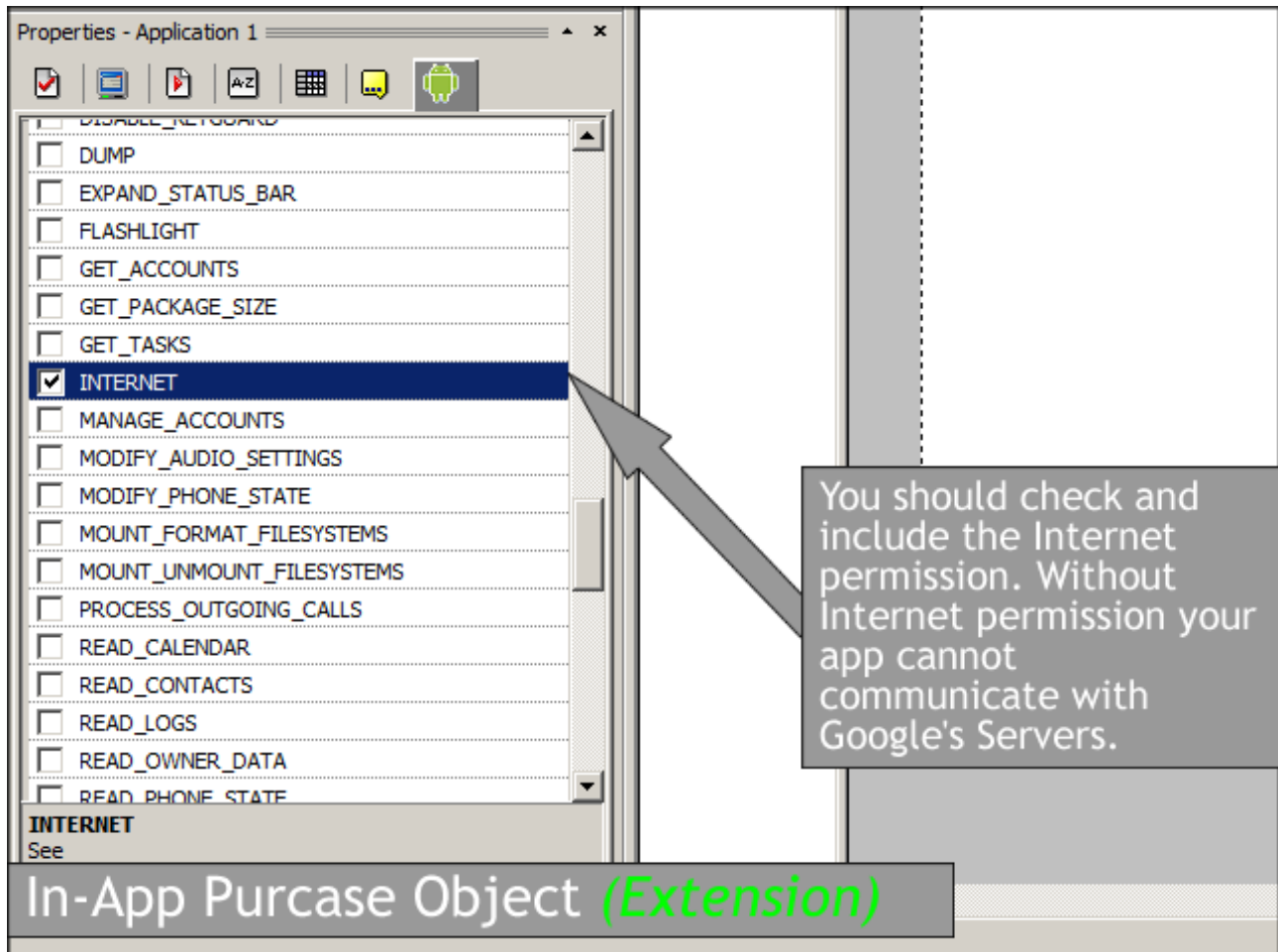


You can put the InApp Purchase extension anywhere in the frame. It can be inside or outside of the frame, regardless this is a hidden extension that only functions in the background of your game. Nothing is drawn to the screen or the frame with this extension.

There is NO NEED to set a BILLING permission as requested by Google's SDK. Fusion automatically sets the BILLING permission up in your Manifest, so no need to worry about that.

## Fusion: Setup App in Play

Now, you will need to set the Internet Permission up for your application. Without the Internet Permission selected, your application will not be able to communicate with Google's Servers.

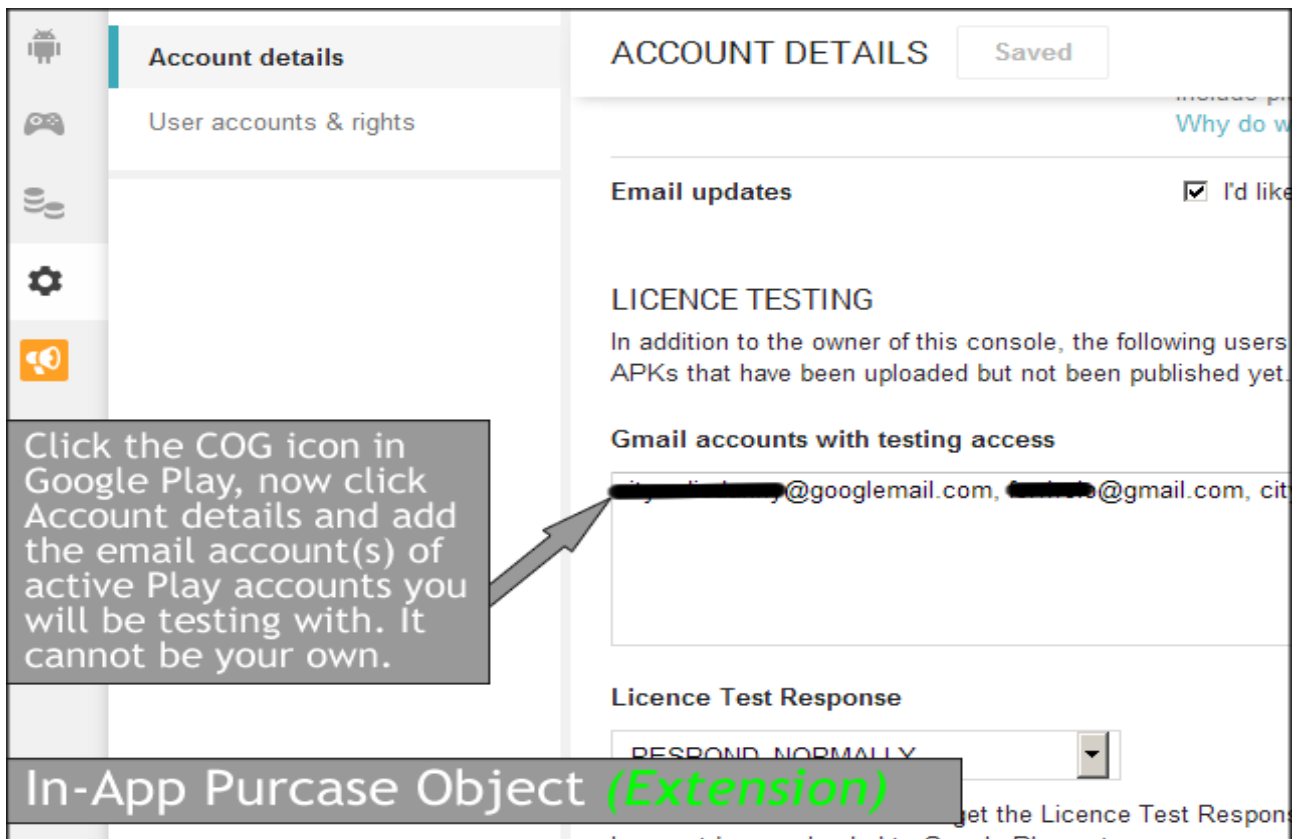


Now, you will need to setup your application inside of Google Play! This is pretty straight forward (although it may appear slightly daunting) I guarantee once you've managed to successfully implement IAP's the first time, you'll be away in creating them for all your future apps and games made with Fusion with no hassle.

Now, you will need to setup inside your Play account, test user accounts. Google will not allow you to test In-App Purchasing with your own developer account. So you have two options here, you can either use someone else's Play account (with permission of course) or you can setup another Google Play account (*Not Necessarily Developer*) just a Google Account that has a credit/debit card attached to it.

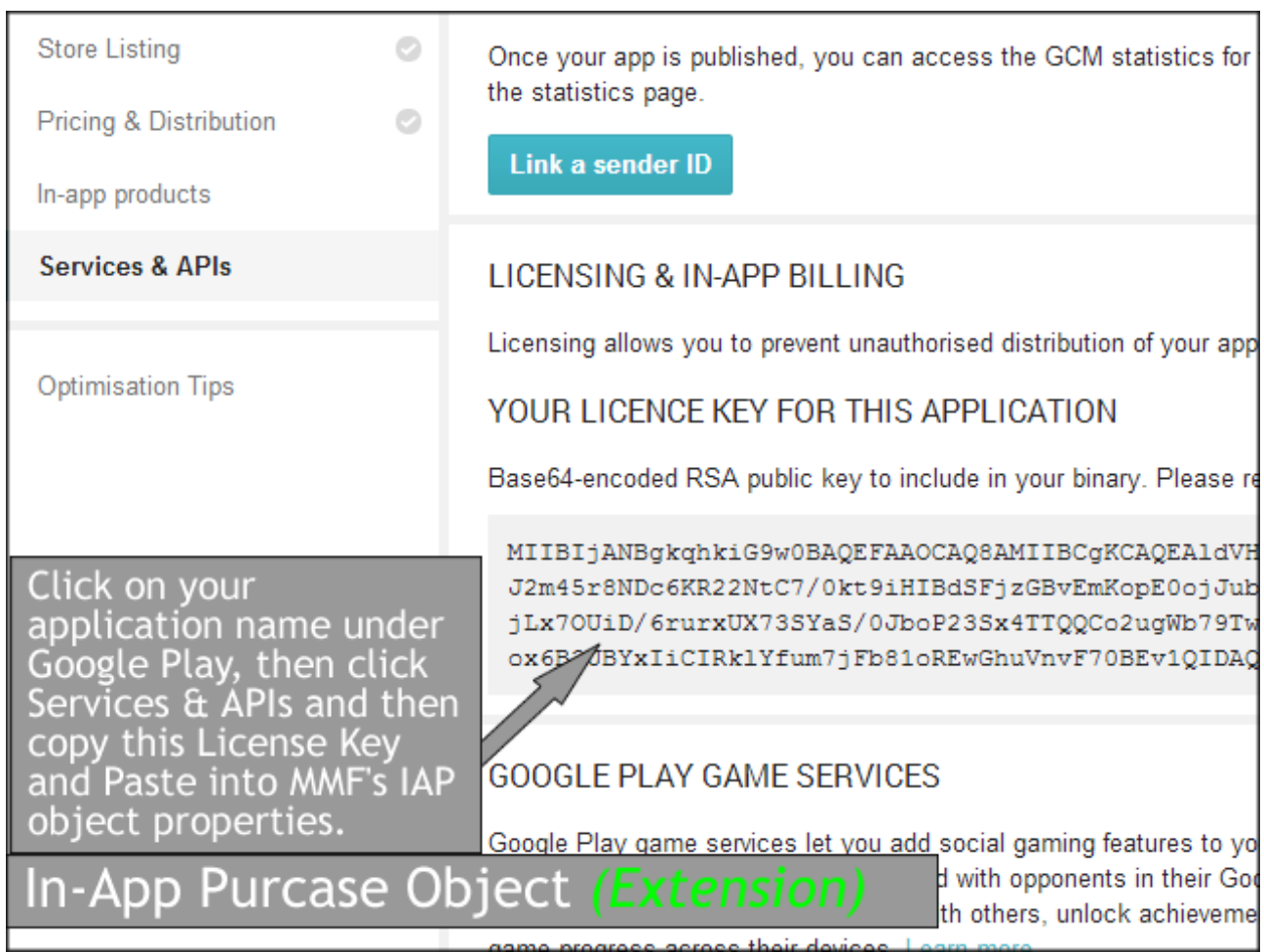
Because I am testing on one of my devices, I use my personal Gmail account (as the primary device account) and it has a debit card attached to it, so I can just build & run and instantly test on my devices, once you have performed this next step...

# Fusion: Setup App in Play



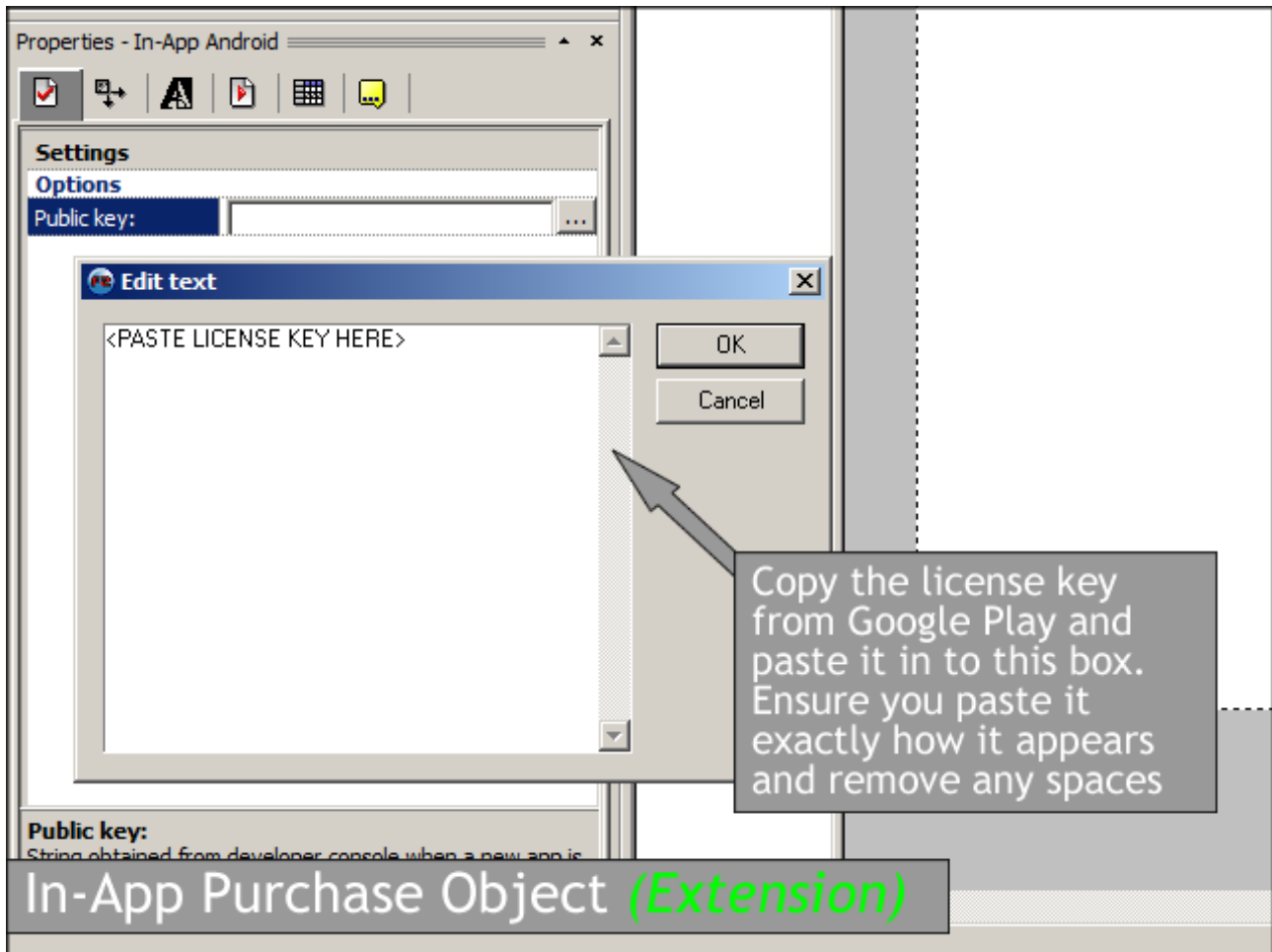
The screenshot shows the 'ACCOUNT DETAILS' page in Google Play. The left sidebar has icons for Android, Game, Accounts, Settings, and a Cog icon. The main content area includes 'Email updates' (checked), 'LICENCE TESTING' (with a note about APKs), 'Gmail accounts with testing access' (with a list of email addresses), and 'Licence Test Response' (set to 'RESPOND NORMALLY'). A grey callout box with an arrow pointing to the email list contains the text: 'Click the COG icon in Google Play, now click Account details and add the email account(s) of active Play accounts you will be testing with. It cannot be your own.' At the bottom, a grey bar contains the text: 'In-App Purcuse Object (Extension)'.

Now, click on Services & APIs inside your application and copy-paste the license key from here into the public key property of the IAP object inside Fusion.



The screenshot shows the 'Services & APIs' page in Google Play. The left sidebar has icons for Store Listing, Pricing & Distribution, In-app products, Services & APIs, and Optimisation Tips. The main content area includes 'Link a sender ID', 'LICENSING & IN-APP BILLING' (with a note about GCM statistics), 'YOUR LICENCE KEY FOR THIS APPLICATION' (with a note about Base64-encoded RSA public key), and 'GOOGLE PLAY GAME SERVICES' (with a note about social gaming features). A grey callout box with an arrow pointing to the license key contains the text: 'Click on your application name under Google Play, then click Services & APIs and then copy this License Key and Paste into MMF's IAP object properties.' At the bottom, a grey bar contains the text: 'In-App Purcuse Object (Extension)'.

## Fusion: Setup App in Play



For some odd reason, Google Play! Forces you to upload your APK BEFORE adding or testing IAPs. However, this is fine, you can just upload your application as it currently is. Just set your game to RELEASE MODE, fill in the details and build an APK in release mode.

Once you have completed all of the above then upload the APK to your Play listing as an ALPHA or BETA release. This APK is just a testing APK so be sure to only use Version 1 and Version 1.0 string in the Android properties.

***Just a sidenote here, you do NOT need to reupload the APK to the store when you make any changes inside Fusion for testing, just keep it in release mode in Fusion and just hit Build & Run to your device. All Google Play is looking for (in test stages) is that the App Identifier and license key are the same.***

Okay, so now we should have the APK setup on the Google Play store and we should also have the MFA setup in Fusion to correspond to all the correct settings, now we need to implement some InApp Purchases for your users!

## Fusion: Design

You will need to design a layer in your frame that will handle the InApp Purchases. It's best to design it in a separate layer so you can keep it away from the rest of your application/game objects. It also means once you've finished designing it you can hide the layer in Fusion so you don't have to keep seeing it or referring to it (unless you need to).



Ideally, you should just have a splash screen that pops up enabling the user to select something to purchase and it proceeds to purchase the object. This is a very basic and simple demonstration just for this guide. Once you have mastered implementing IAPs of course you can expand how your app/game looks and how you present your IAP's to your users.

Your method of displaying the purchase screen can be as simple or as complex as you like, however there are a few rules you must follow. These rules are:

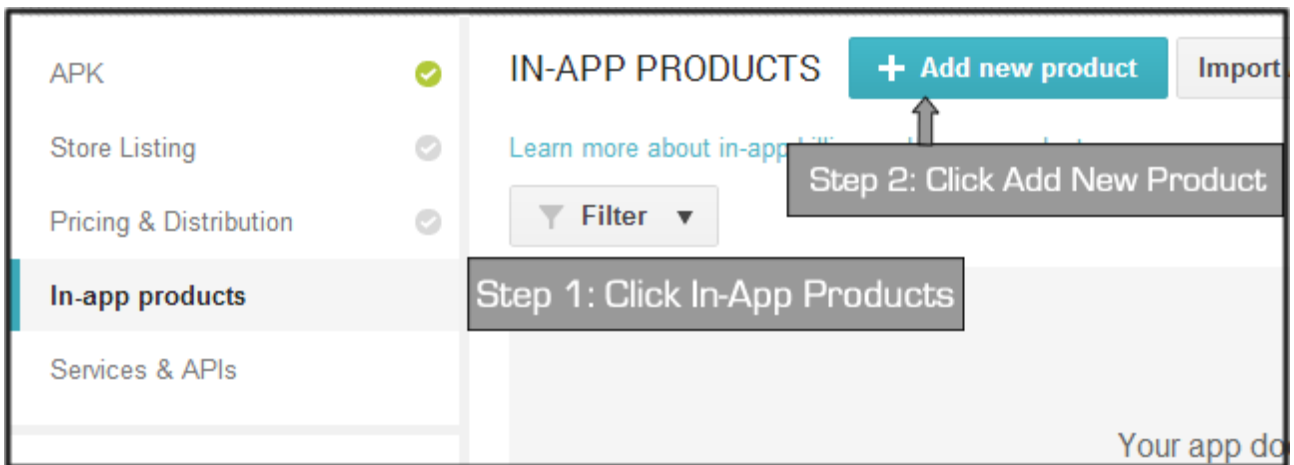
- 1) The user must be able to clearly see what he/she is purchasing
- 2) The user must be able to clearly see pricing and availability
- 3) There should be a button or indication on where to click to buy an IAP

This is pretty much the design section complete. Let's hop over to the implementation section where we get our hands dirty with putting events into your MFA, from start to finish!

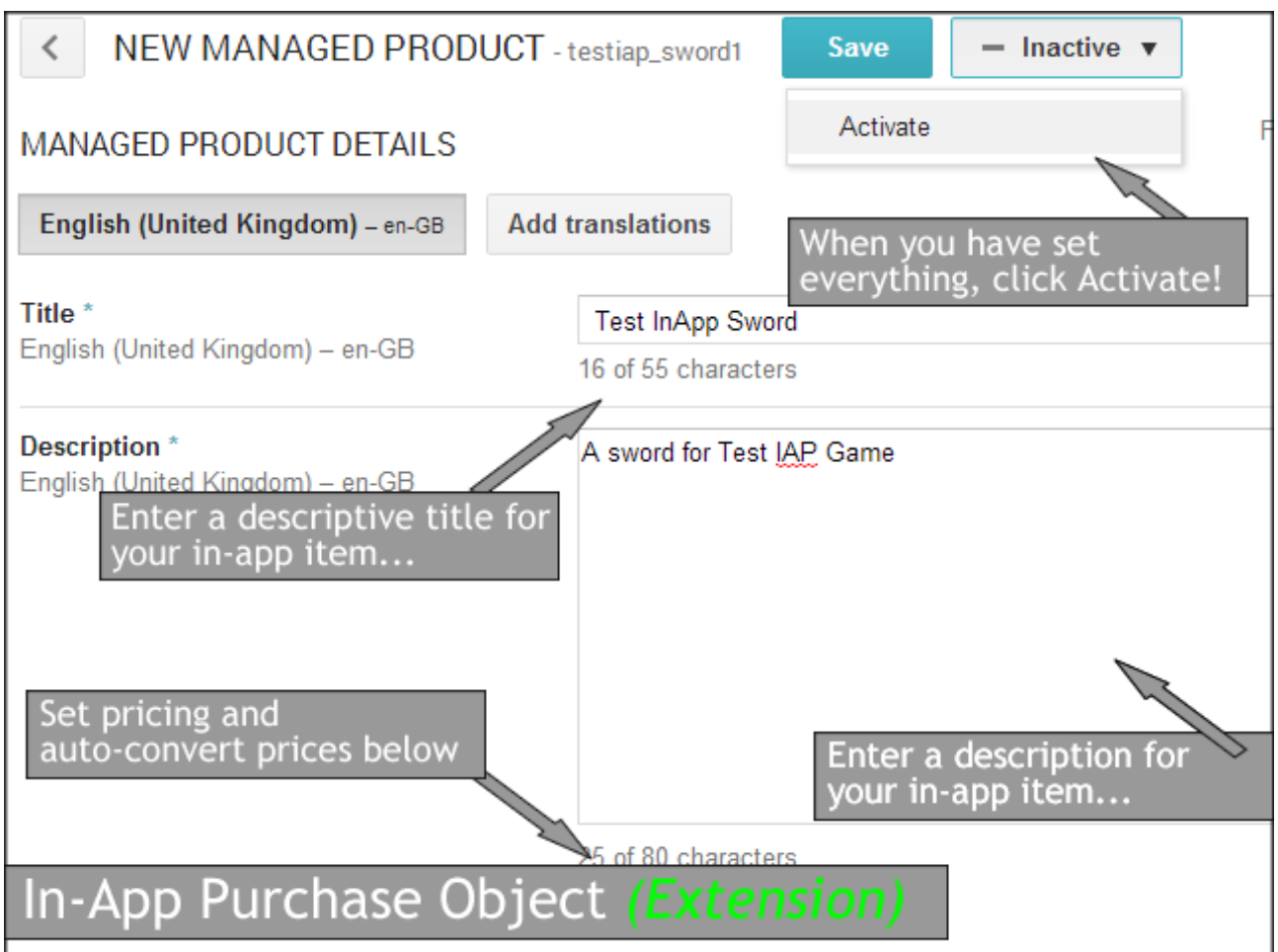
# Fusion: Implementation

Google's documentation isn't the easiest to read, understand and/or follow, hence this guide. I hope this guide up until this point at least is readable and understandable thus far. So, now we get to the nitty gritty and one of the final steps of IAP implementation in your Fusion App/Game.

Ok, so let's jump back to Google Play and setup the items that will be available for purchase in your game.



Once you have clicked Add new product, you then click Add Managed Product, give it a unique name such as testiap\_sword1 then click Submit. You will then be presented with a screen to give more information about that item such as title, description and price. Once you've set all that up, click Activate!



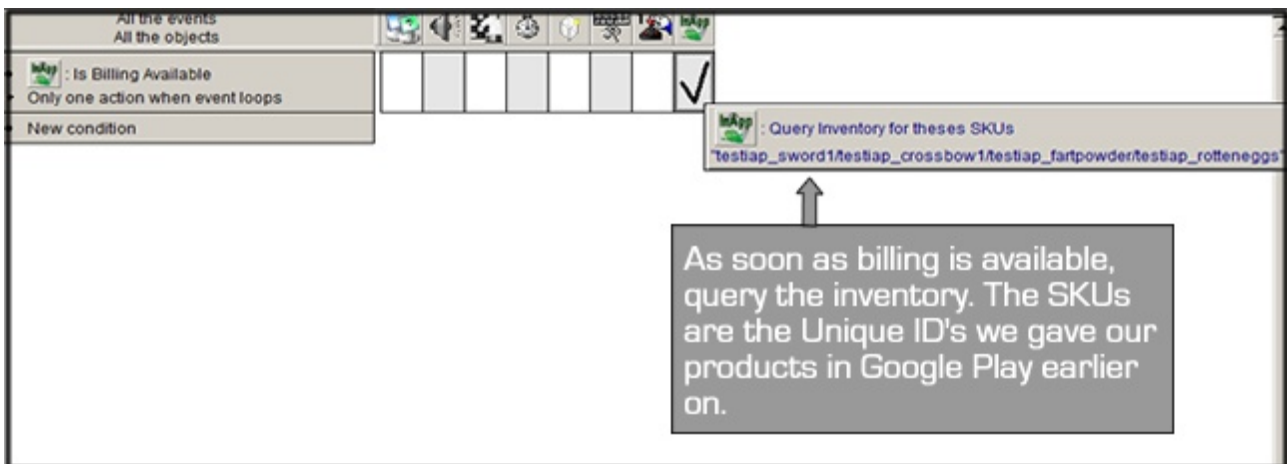
# Fusion: Implementation

You can repeat the above step for the other items listed here in this guide in the design section above (*ie: **Fart Powder / Rotten Eggs / Crossbow***). ALWAYS make a note of the UNIQUE ID you give to your IAP's. You will need this unique ID to connect Fusion to Google to tell Google which product to purchase.

Okay, now finally onto Fusion. Jump back to Fusion and your app/game and let's implement some startup events. First we'll need to check that **Billing is Available**. I prefer to do this at the Start of the Frame as it enables me to query the inventory and get it all out of the way as soon as possible. You can however, implement this anywhere. Anyhow, onto that later on. Let's start of with one simple event:

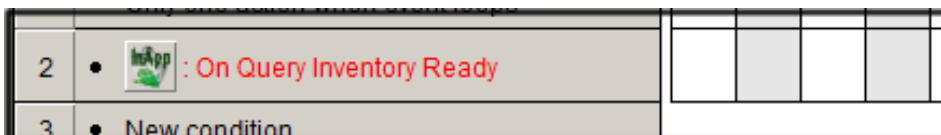
## Is Billing Available?

+ Only One Action when Event Loops



You will need to ensure you also attach “Only one action when event loops”, because if Billing is Available this action will loop constantly (not necessary and will hog performance).

So, we've told our App to QUERY Google's Servers, in other words, LOOK at our account and check that those In-App Items are indeed available for our users to buy. Now we need to detect that the query has happened:



This is an immediate condition which means the very split second the inventory is ready to be read by our app, it will trigger. So what we do now is, set a fastloop to run when this is triggered. **Wondering how many times to run the fastloop? See Next...**

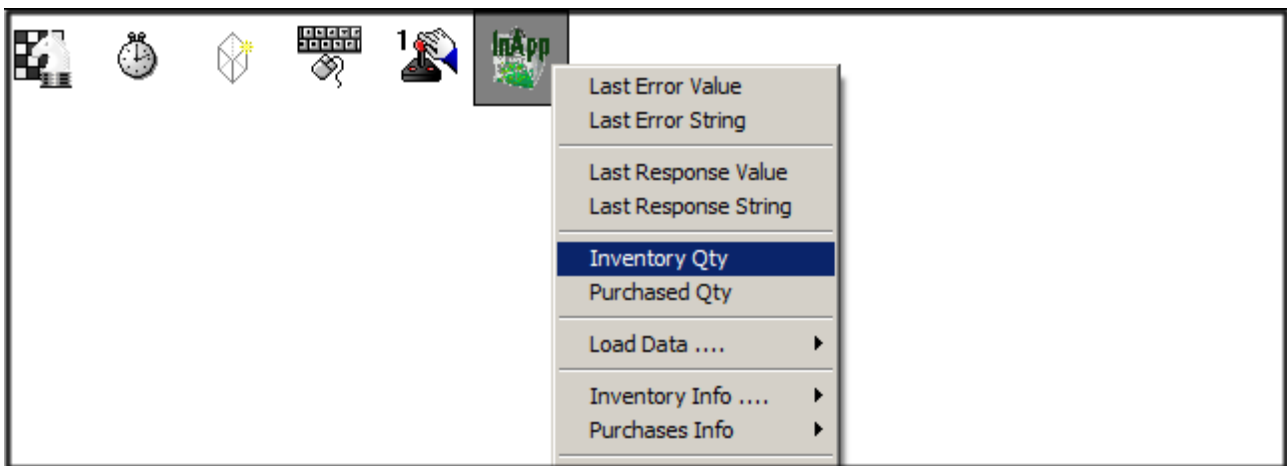


# Fusion: Implementation

You will need to run the fastloop the same amount of times there is objects available. This is so we can loop through all the objects in sequence. So, for example, if there is 5 in-app items available, we need to loop 5 times. Rather than inputting a static number, we'll retrieve this quantity information via the inventory we just queried.

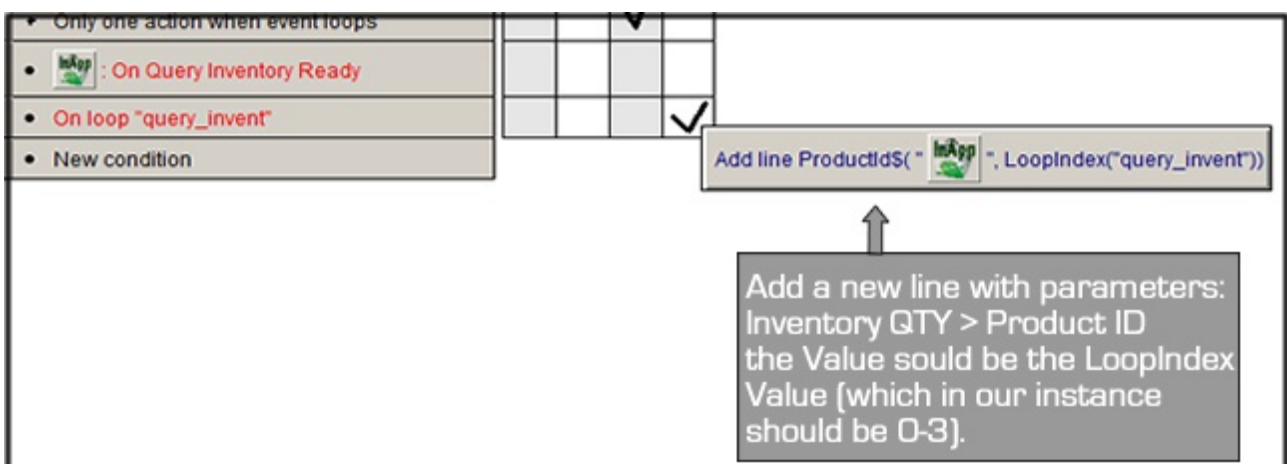


Click, Retrieve Data from an Object. Then right-click on the In-App object and select Inventory QTY. This is the quantity of inventory available (it was queried above). We will call this fastloop "query\_invent".



You should end up with something similar to this in the expression editor:  
`InventoryQty( "In-App Android" )`

Now, this fastloop will run over the inventory. On each fastloop, you could add a line to a list object (in your frame) so you can visually see on the first test, that it is indeed returning the correct items back. So let's do that. Insert a list object into your frame and let's do:



# Fusion: Implementation

So, we have started a fastloop upon Inventory Ready. We use “Inventory Ready” as a trigger condition to tell our app that our inventory (on Google's Servers) is now accessible and is ready to be looped. So we call a fastloop which does this:

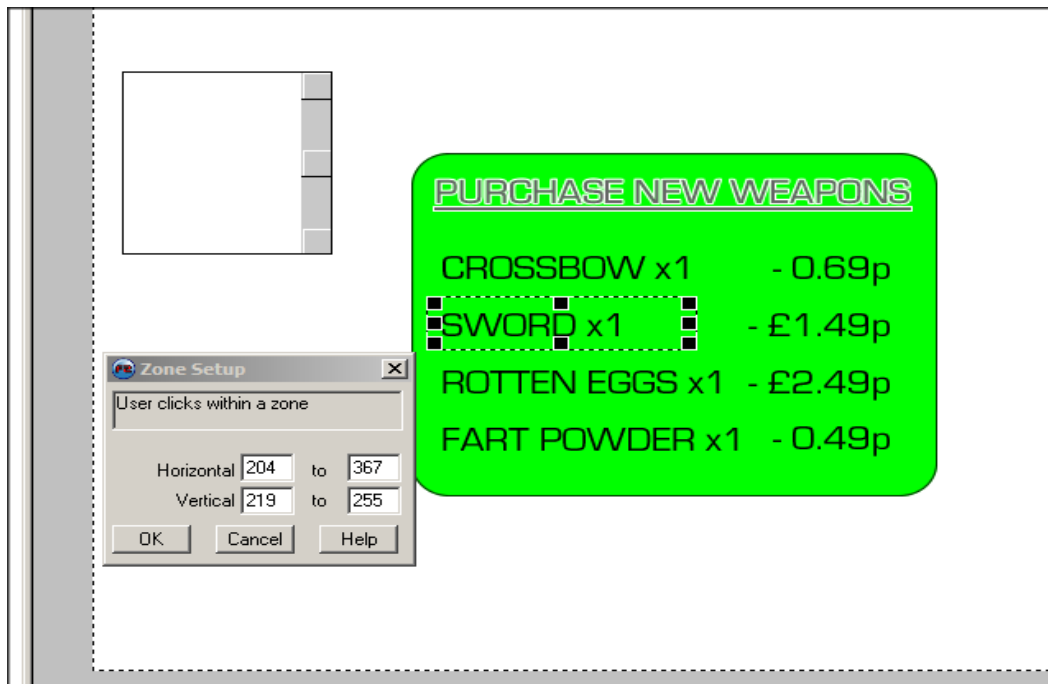
**OnLoop:** Add a new line to list box: “product\_id\$” and the value is the 'actual' product id according to the loop.

Once the fastloop has finished looping through our available inventory on Google's Servers, the list should now be populated with the SKU's (Unique ID's) we gave our in-app items earlier on.

Now is the time to do a test on your Android device(s). Attach it to your computer via USB and hit Build & Run. Some people prefer to build the APK and transfer it manually via DropBox, FTP or file transfer. Either will work. Once you run your application on your device, the list box should populate itself with lines of product names available in your Google Play In-App List. If this works, then everything is fine so far. If not, then you may need to read over the steps above again (*Also ensure you have an active internet connection available on the device*).

Once you have completed this step and it's working, let's move onto purchasing and consuming these in-app items.

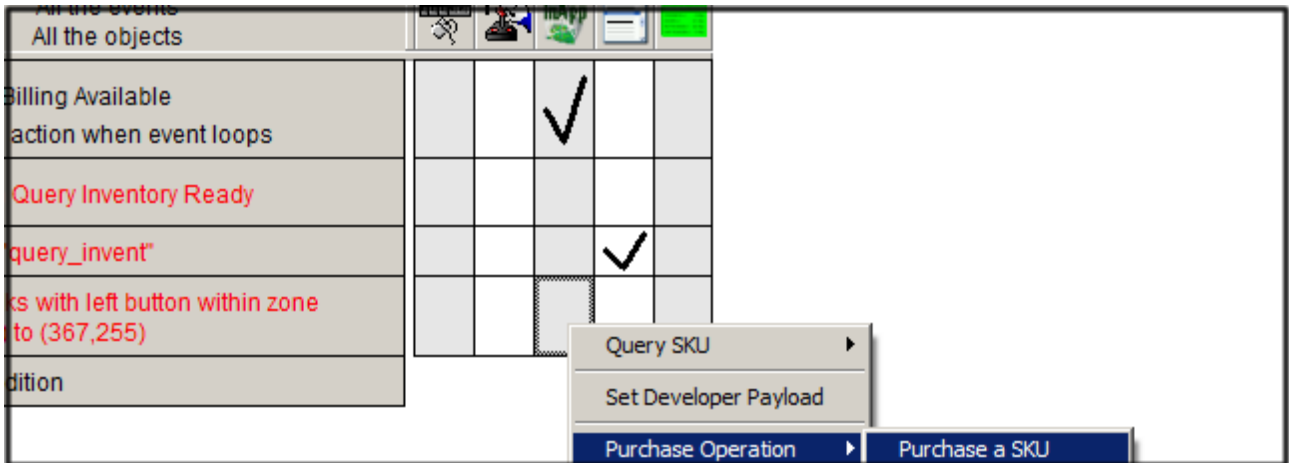
Let's do a new condition, asin the user has clicked on Sword1, so let's purchase it. First we need to create a condition where the user clicks on Sword1:



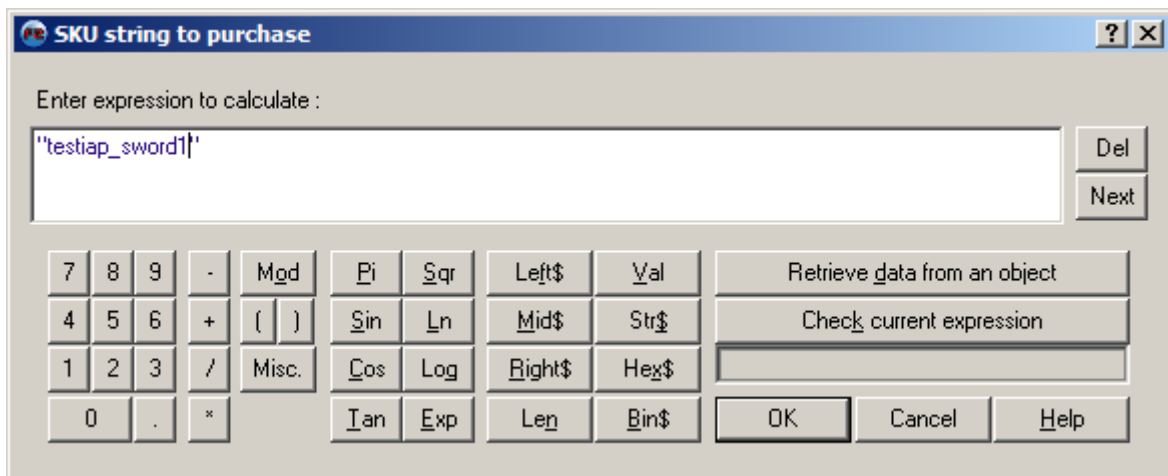
We'll setup a new condition, “User Clicks within a Zone” and set the zone up over Sword x1. Click OK.

# Fusion: Implementation

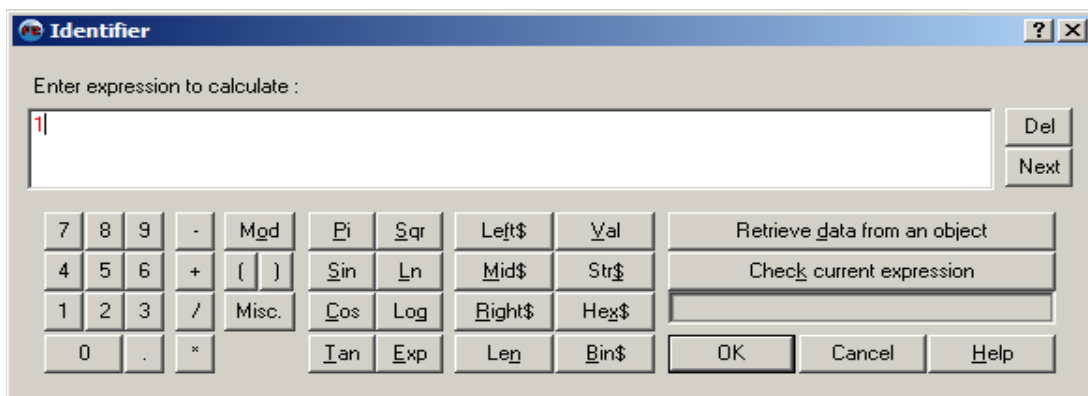
Now, we need to tell the IAP object to purchase Sword 1, which should have an SKU (unique id) of testiap\_sword1, so let's do that:



Right-Click under the IAP object and select Purchase Operation > Purchase a SKU. In the dialog it will prompt you for the SKU of the in-app item to purchase, in this instance it is sword x 1 (SKU: testiap\_sword1) so go ahead and input that:

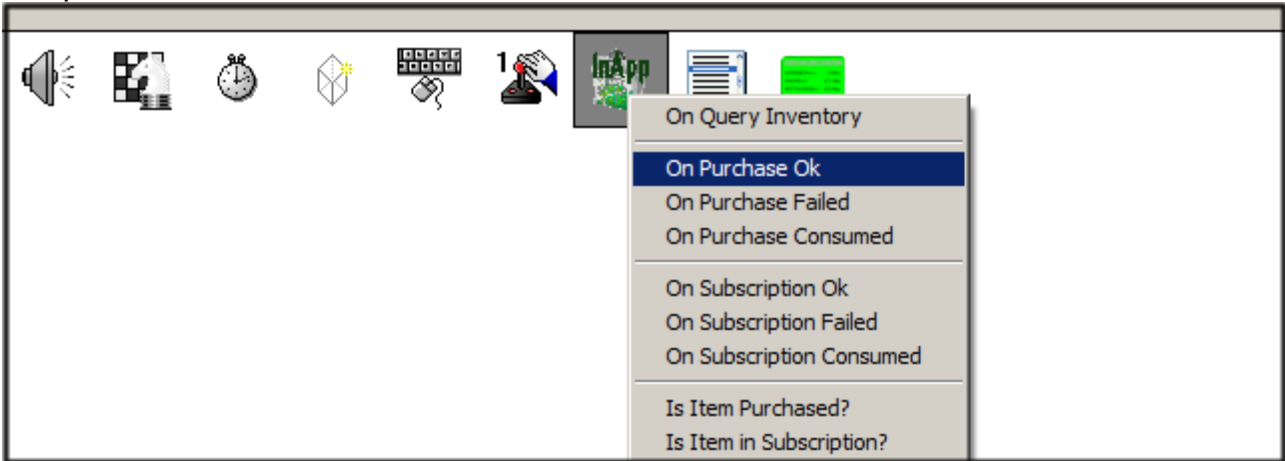


It will then ask you to provide an identifier, this should be a numerical value. So for this guide we will use 1 for the sword, rotten eggs should be 2 etc.

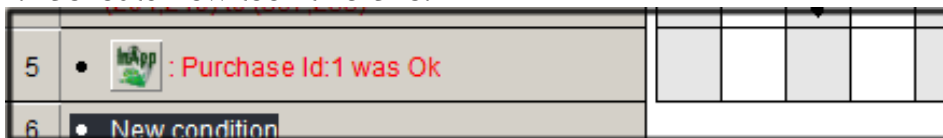


# Fusion: Implementation

Now that we've done that we need to check that the purchase went through OK, so let's setup a new condition:



When you select this it will ask you for an identifier, we used 1 for the sword, so put 1 and Click Ok. It should now look like this:



Once the purchase is OK, we need to Query the inventory again, you should always query the inventory after every purchase and everytime the user consumes an item. This just keeps your application in-sync with what the user CAN purchase, what the user has ALREADY purchased and what the user has in his purchase inventory.

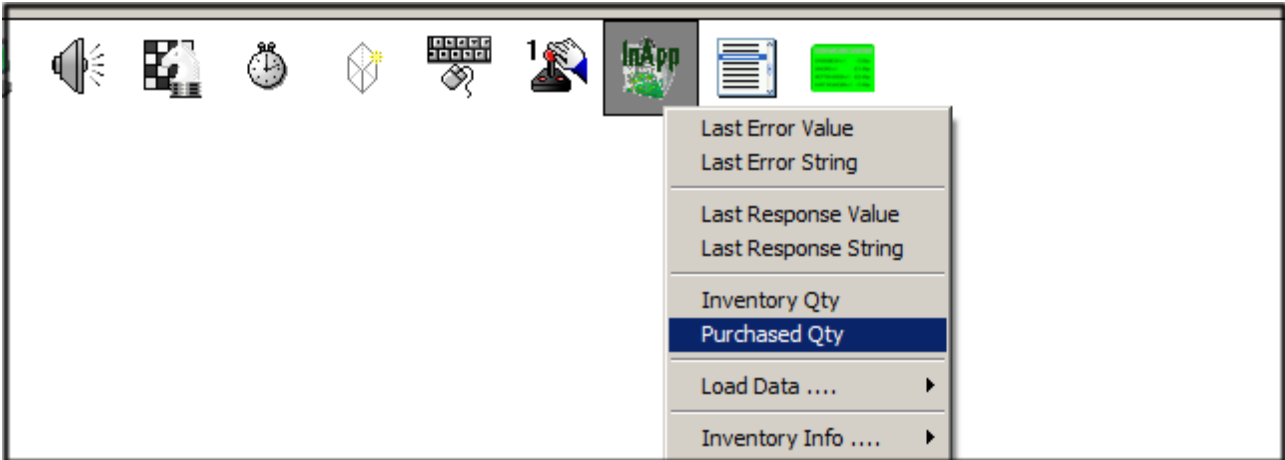
**Note:** You essentially have two inventories-in-one. The first inventory is what is available to purchase (ie: in-app items) the second is the users inventory (what the user has already purchased). Think of the second one like a backpack containing items the user has already purchased.

Now, we don't need to do anything with QUERY INVENTORY action as we already took care of that above with the fastloop. However, you might want to check what the user has purchased, if this is the case, then add a second list object to the frame and jump back to the second condition we did:

“On Query Inventory Ready”:

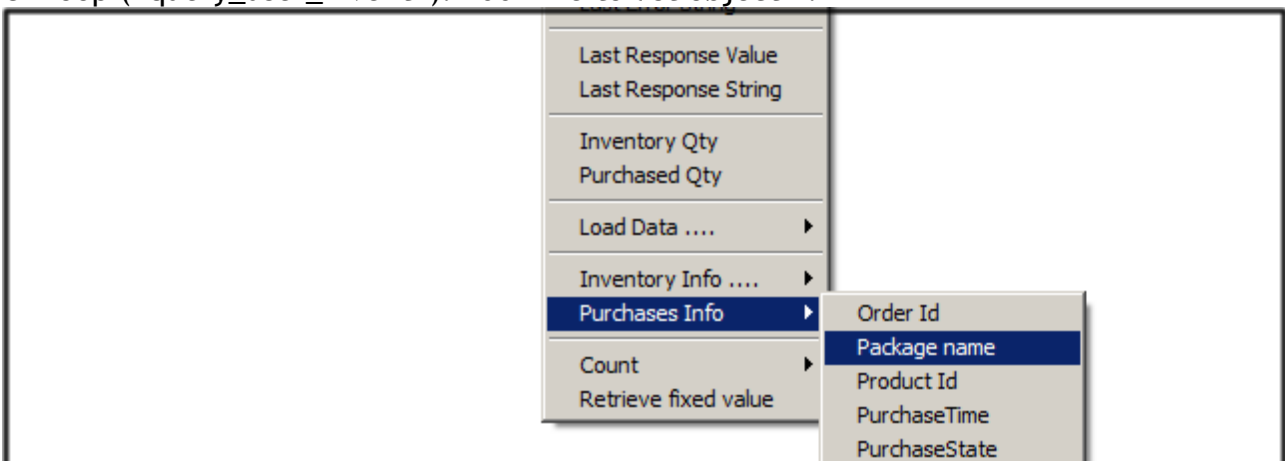
We need to start a second fastloop after our first fastloop. Let's call it 'query\_user\_invent'. The amount of times to loop should be:

# Fusion: Implementation



So, it will loop given the amount of quantity in the Purchased inventory. Then:

OnLoop (“query\_user\_invent”): Add Line to list object 2:



Choose Product ID. When it asks for a value, type in `LoopIndex(“query_user_invent”)` so it should now look like this in the expression editor:

```
PProductId$( "In-App Android", LoopIndex("query_user_invent"))
```

This will now populate the second list object you put in the frame with all the items the user has purchased.

So, now you should have two lists, one for the inventory with available in-app items to buy and a second list for the inventory of what has been purchased by the user. Now, you can choose when to consume the item, let's say for example the user selects SWORD from their backpack in the game, they have now USED the item in the game and it's available for them to purchase again if they want so let's consume an item.

So, let's pretend the user clicks on the sword icon in-game. You want them to USE the sword item they purchased, so setup an event where they click this icon and on the action, right-click on the IAP object and select Purchase Operation > Consume a SKU.

# Fusion: Implementation

You will want to CONSUME the Sword1, so let's put in its SKU (unique ID):

With identifier (1 - from earlier):

Click OK.

Now, we obviously need to check with Google that this consumption went through OK and Google's Servers have updated that this item has now been consumed so let's do:

5	•  : Purchase Id:1 was Ok																			✓	
6	• User clicks with left button within zone (320,378) to (496,432)																			✓	
7	•  : Purchased Id:1 was consumed																				
8	• New condition																				

What you do after the item is consumed is entirely up to you, it will disappear out of the list of what the user has purchased. So in this instance, you would equip the player with

## Fusion: Implementation

the sword he purchased. Because it has now been consumed, it means it will now become available for the user to purchase again in the future.

You can test at any time again on your Android device. Don't forget when you consume or purchase an SKU (in-app product) you must always trigger the action "Query Inventory" so it keeps the lists up to date with Google's Servers.

Any test purchases made whilst in test mode will not deduct payment. Please refer to Clickteam's Official Help-File and Example MFA's that came with the Android Exporter.

That's it! The same principal above applies for both Managed and Non-Managed Products. Managed means Google will keep track of all the orders and inventory, non-managed means once the purchase has been made, Google no longer tracks the item so you must do this yourself by saving the data locally (ie: if user buys sword1 when it's an unmanaged IAP, if they exit the game and re-enter there won't be sword1 in stock with Google. You would have to save this information to an INI or an array so the user has the sword when he closes and re-enters the game).

## Fusion: FAQ's

### FAQ Frequently Asked Questions

#### **Is it better to use Managed or Un-Managed products?**

It is entirely your call. Managed means Google will store every last detail about the order, the product and quantities of products. Un-Managed means the user can still purchase but all logging stops after the purchase. You must track purchases yourself in the app.

#### **Developer Payload? What is it?**

The developer payload is a kind of security string sent to Google's Server (by you) that will be random and unique for that and every purchase in your game. I didn't include it in the above guide because I wanted people to figure out how to implement IAPs without the added jargon of a Payload. However, in a nutshell, you will need to include a payload somewhere, so at the Start of the Frame you could generate a series of random numbers. Just do: Start of Frame: Set Developer Payload.

#### **What is random though when two could potentially collide?**

This is true. I have worked with a lot of security techniques over the years and even though things are random, you could still get the same output twice which could confuse things. I have figured a way around this. Each Android Device has a UNIQUE DEVICE ID. You can retrieve this from the Android Object. So I set my developer payloads to UniqueDeviceID+(10+Random(383748))+TimeDate\$ Randoms. I will cover this in another guide or video on Udemy.

## End Note

Thanks for downloading this PDF. I hope it was of use to you. There are only a few other conditions, actions and events related to the IAP object. It comes with its own help file. You can check this out but they're pretty self-explanatory such as "On Purchase Failed" which obviously means the purchase failed. If this condition crops up then you could set a string to retrieve the last error message from the IAP object etc.

I will be providing more information via my Udemy course which you can find here:  
<https://www.udemy.com/creating-games-in-multimedia-fusion-2-beginner-course/>

You can find a wealth of information over at the Clickteam Community Forums too.

I hope this Guide was useful and I hope you managed to follow it successfully. Good luck in creating your games for Android and making money from In-App Purchasing.