Google IAP and BrainCloud

1. Open your Unity project.
2. Get the latest brainCloud lib here and add it to your project https://github.com/getbraincloud/braincloud-csharp/releases
3. Follow this tutorial for information on how to activate Unity IAP, setup your project to use Unity Services and to integrate google play billing into your project:

   https://developer.android.com/google/play/billing/unity

TIPS

- This tutorial shows you in-depth steps for setting up Unity IAP, and gives you an example of creating a Purchasing Script in Unity to properly use the Unity IAP. Refer to this or our braincloud example if you're having trouble.

  https://learn.unity.com/tutorial/unity-iap#5c7f8528edbc2a002053b46e

- note that instead of the StandardPurchasingModule for your configurationBuilder :

  ```
  var configurationBuilder =
  ConfigurationBuilder.Instance(StandardPurchasingModule.Instance());
  ```

  You will want to use google's

  ```
  var configurationBuilder =
  ConfigurationBuilder.Instance(Google.Play.Billing.GooglePlayStoreModule.Instanc
  e());
  ```

Next you will need to follow this tutorial to properly set up your app in order to create products for your app's store in the google play console. Some parts of it may be a little outdated, so we will list out the steps we took.

https://docs.unity3d.com/Manual/UnityIAPGoogleConfiguration.html

1. Go to your app in the google play console. If you do not have one, make one!

2. Go to Store Presence > Main Store Listing and fill in the blanks and images for your store listing

3. Go to Dashboard and make sure to do all the initial app setup on Dashboard. This will allow you to make a test release

4. Before adding an in-app purchase, it may ask you to first upload an apk. Note that the APK needs to be at least API level 29 to comply with Google's restrictions. It also has to be a build including ARM7 and ARM64 architecture. The apk will also have to be signed with a keystore. Each new upload needs a new version number as well.

Tip for Android API Level

- Go to your Android SDK Manager (usually easily found using Android Studio)

- Make sure you have SDK 29 or above downloaded and note the file path for Android SDK Location. Navigate to the file and under the platforms folder copy the android-29 or higher folder

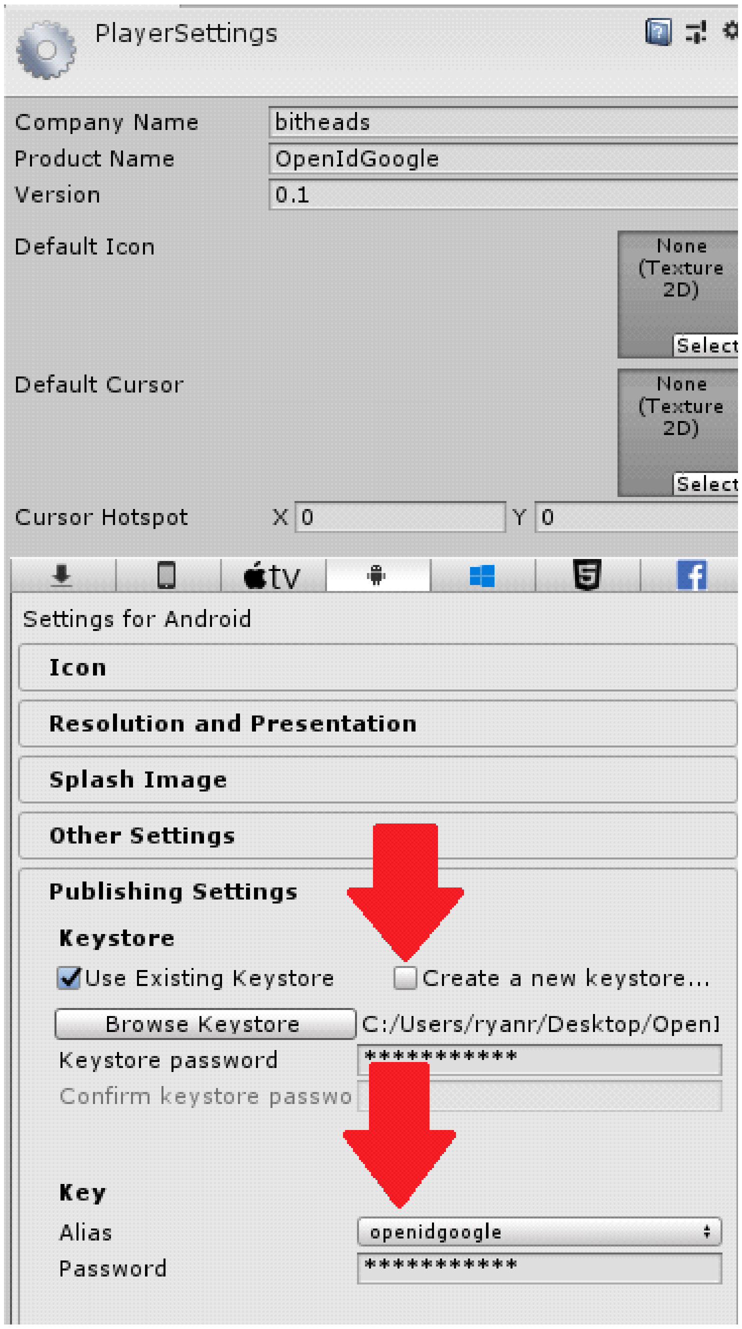
- Go into Unity, Edit > Preferences and in External tools note the path of the Android NDK folder Unity uses. Navigate there go into platforms and paste the android-29 folder into that folder. Then when building, just make sure that in Player Settings, the Target API Level is highest installed.

Tip For ARM7 and ARM64

In Build Settings > Player > Configuration > Scripting Backend, make sure to change Mono to IL2CPP. You may need to add this support to your Unity if you did not download it with IL2CPP support.

Tip For Keystore

If you don't already have a keystore for your app you can create one in Unity. Make sure to personalize your Company Name and Product Name in Build Settings > Player, and use those credentials to update your package name in Build Settings > Player > Other Settings > Identification before creating your keystore. DO NOT LOSE THE KEYSTORE. Once you upload the apk to your app on Google Play console, you will not be able to sign new releases with anything BUT that keystore.


PlayerSettings

Company Name	bitheads
Product Name	OpenIdGoogle
Version	0.1
Default Icon	None (Texture 2D)
Default Cursor	None (Texture 2D)
Cursor Hotspot	X 0 Y 0

Settings for Android

Icon

Resolution and Presentation

Splash Image

Other Settings

Publishing Settings

Keystore

✓ Use Existing Keystore ☐ Create a new keystore...

Browse Keystore C:/Users/ryanr/Desktop/OpenI

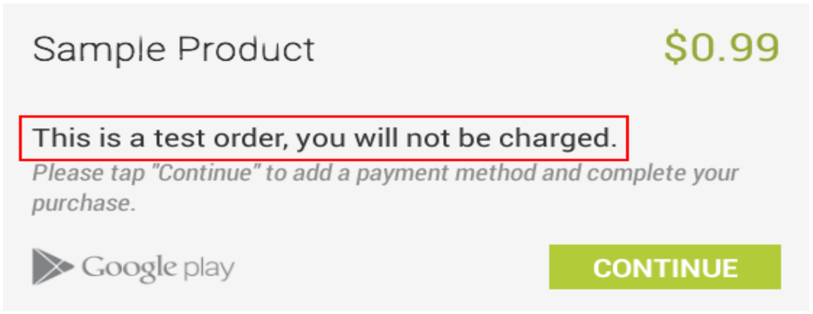
Keystore password ***********

Confirm keystore passwo

Key

Alias openidgoogle

Password ***********


Sample Product $0.99

This is a test order, you will not be charged.

Please tap "Continue" to add a payment method and complete your purchase.

Tip: In order to test the product you must set up your testers both in Test Track(Open,Closed or

Internal) and License Tester in your Developer Account settings in Google Play Console.

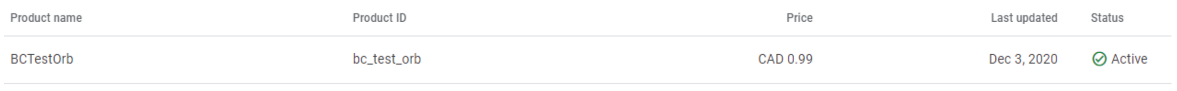
- For purchases from license testers, a purchase will be refunded after 3 minutes if your app does not acknowledge the purchase and you will receive an email about the cancellation. You can also check the **Orders** tab in the Google Play Console to see if an order was refunded after 3 minutes.

IMPORTANT: DIALOGUE SHOULD STATE THIS IS A TEST ORDER, OTHERWISE TESTER WILL BE CHARGED WITH REAL MONEY.

Go back to Testing > Closed Testing and set up a release. Upload a signed, non-development build android apk as mentioned in the unity documentation. Manage the track you added, and create a new closed testing release. It will likely take a couple of days for Google to review your release. In the meantime you should be able to set everything else up.

5. Once you get an apk uploaded you should be able to add an in-app purchase. Add an in-app product under Monetize>Products>In-app products. Fill in the blanks, and take note of the ProductId, that is something you will need for brainCloud and in your Purchasing Script. SAVE IT. You may need to set up a monetization account, the google play console will guide you through this.

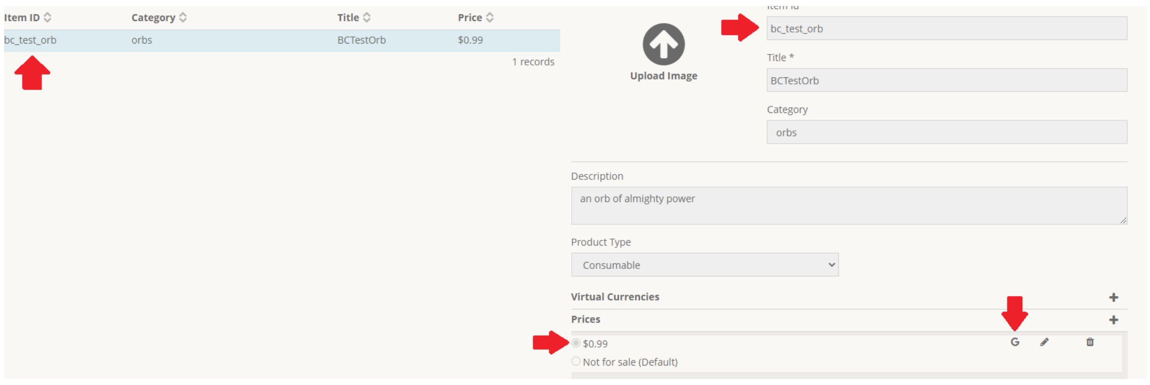
Set a price, the minimum price is 0.99$, Don't worry you testers will not be charged anything when they attempt a purchase.



Product name	Product ID	Price	Last updated	Status
BCTestOrb	bc_test_orb	CAD 0.99	Dec 3, 2020	✓ Active

Time to go to your brainCloud app! If you haven't already made one, make one now! Be sure to add the platform Google Android in Core App Info > Platforms

Go to MarketPlace > Products and add a product. MAKE SURE THE Product Id matches the product id of the in app purchase you made in google play console. Make sure to match the prices and that the price is enabled for the Google Play Product Id.



Item ID	Category	Title	Price
bc_test_orb	orbs	BCTestOrb	$0.99

1 records

bc_test_orb

Title *

BCTestOrb

Category

orbs

Description

an orb of almighty power

Product Type

Consumable

Virtual Currencies

Prices

$0.99

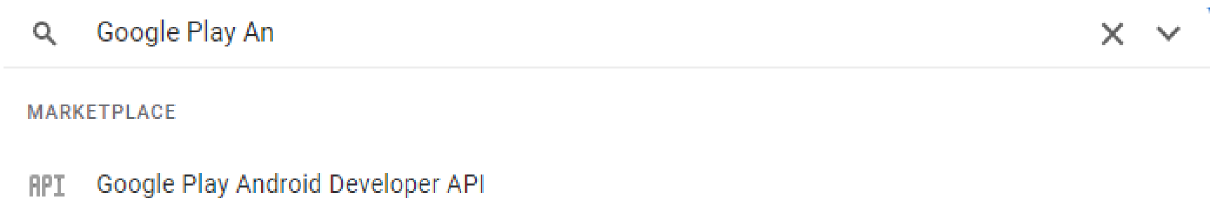
Not for sale (Default)

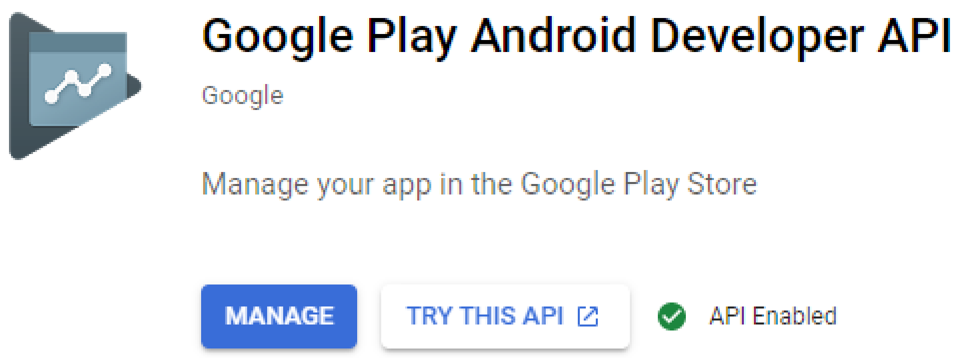
Now we need to make a Google Cloud Project and link your Google Play Console project.

1. Sign into the google account you use for your google play console and create a new app, I recommend for clarity, name the app the same as the app you made in Google play console.

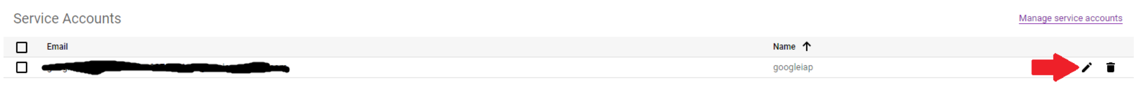
   https://console.cloud.google.com/

2. In APIs and Services First go to dashboard and ENABLE APIS AND SERVICES, and search and enable Google Play Android Developer API.




Google Play Android Developer API
Google
Manage your app in the Google Play Store
MANAGE
TRY THIS API
API Enabled

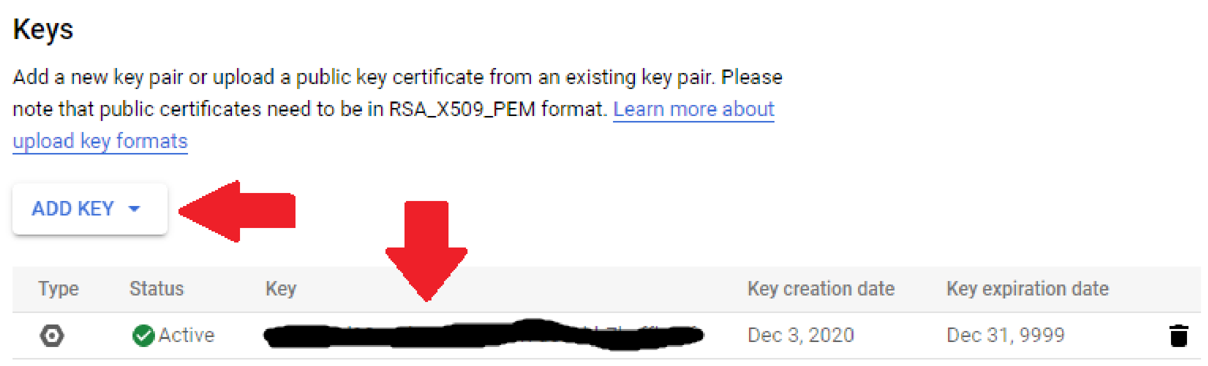
3. Then we need to set up a service account and a p12 certificate for brainCloud to use. Go to dashboard and ENABLE APIS AND SERVICES > Credentials, and create a credential then create a service account. I make the service account an Owner but its optional. Once you've generated your service account you should see it there on the APIs and Services > Credentials page… click on it on the edit pencil!


Service Accounts

Manage service accounts

	Email	Name ↑
☐	[redacted]	googleiap

4. Once you click on the edit, you'll be able to make a key. We now need to add a key, which will make a p12 certificate for us to use. DO NOT LOSE this p12 certificate.



Keys

Add a new key pair or upload a public key certificate from an existing key pair. Please note that public certificates need to be in RSA_X509_PEM format. Learn more about upload key formats

ADD KEY

Type	Status	Key	Key creation date	Key expiration date	
	Active	[redacted]	Dec 3, 2020	Dec 31, 9999	

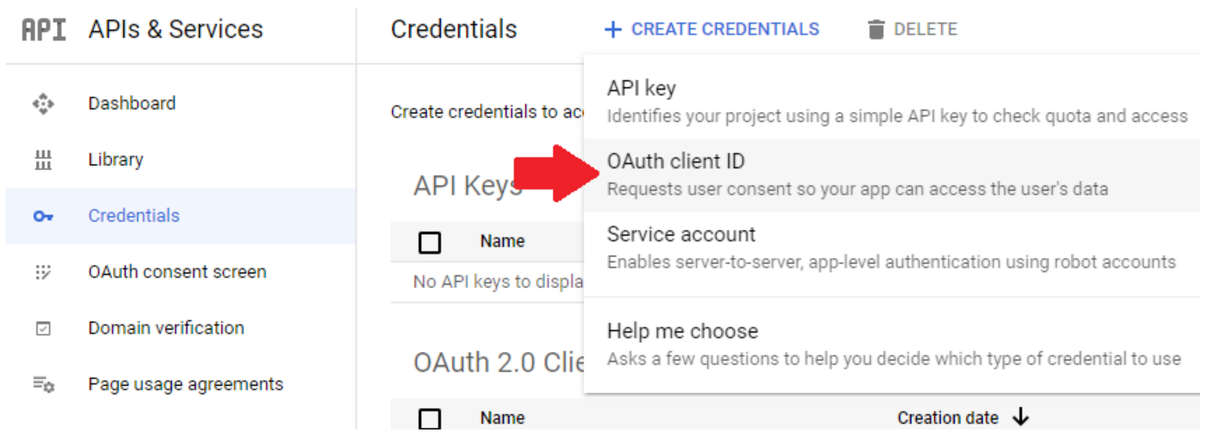
Now we need to go back to Google Play Console and link the project to the project you made on Google Platform, **you will only be able to do this as the OWNER** of the google play console developer account. If you are not the owner, you will not be able to go any further. You need to get the owner to do this if you are not the owner. **Admins do not have the same access as the owner.**

1. In the Google Play Console go to Settings > Developer Account > API Access and Link an existing project. If you do not see the google cloud console app you just created in the list, it's because you did not search and enable the Google Play Android Developer API in the Google Cloud console of your app. (go to dashboard on Google Cloud Console app and ENABLE APIS AND SERVICES, and search and enable Google Play Android Developer API)

2. Find your service account email and grant it access! We give ours Admin as this account will connect through brainCloud for purchases. Go back to your API Access page and you should see your service account.

3. You will need to make an oauth brand for the link to work as well. To do this go to the API Access Page and in Oauth clients Configure Oauth consent screen if you don't have any to choose from.

4. Once you have created an auth consent screen you will notice that you also have to make an Oauth Client. Go back to your google cloud console to do this. Under APIs and SERVICES > Credentials and create credentials. Create an Oauth client Id choose Web Application for BrainClouds purposes. Once created, take the clientId and the client secret we will be adding those to brainCloud settings. Then go back to your Google Play Console and refresh your Oauth clients, and confirm it shows up there.


API APIs & Services

Dashboard
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements

Credentials + CREATE CREDENTIALS DELETE

Create credentials to ac

API Keys

Name

No API keys to displa

OAuth 2.0 Clie

Name Creation date

API key
Identifies your project using a simple API key to check quota and access

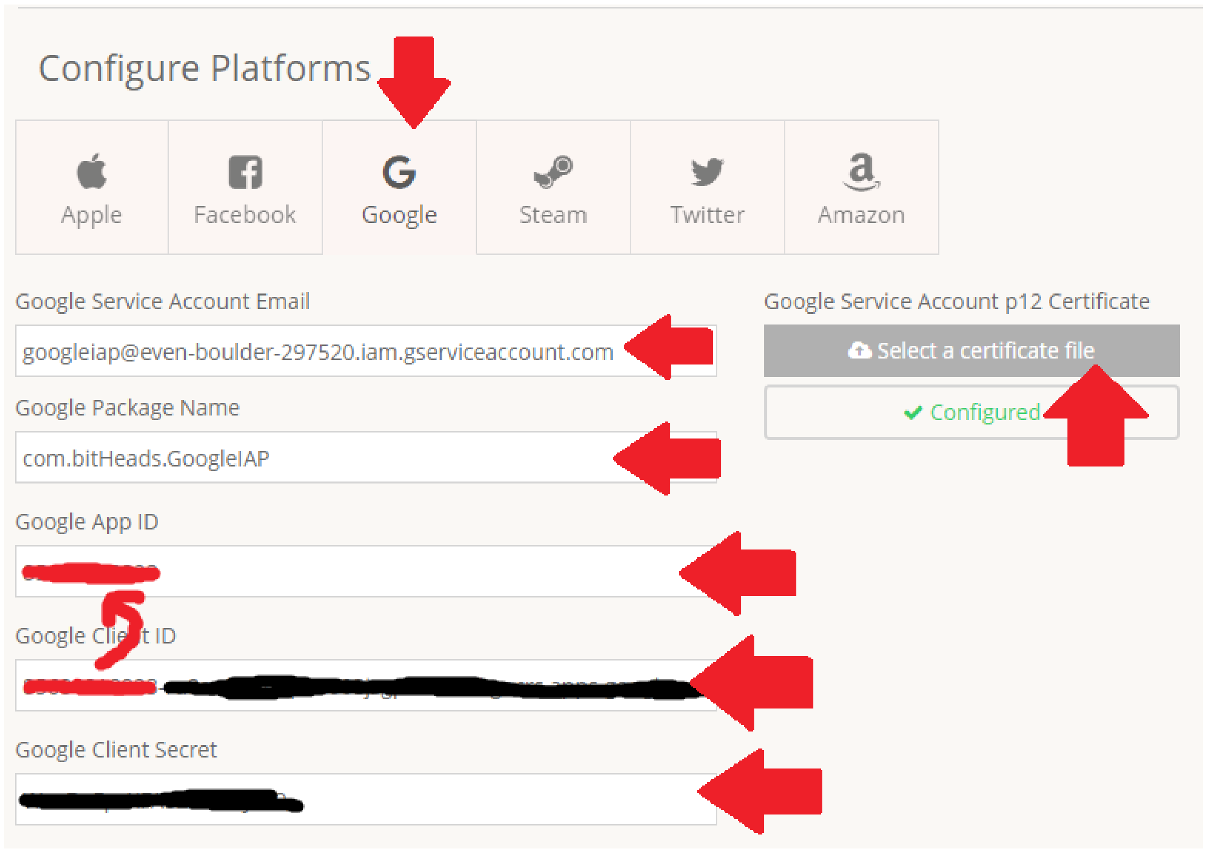
OAuth client ID
Requests user consent so your app can access the user's data

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

Now we need to go back to your brainCloud app!

1. Go to Core App Info > Application IDs and notice the G for google in Configure Platforms. Add your service account, and p12 certificate to the fields there. Then your Google package name should be the package name of your app. And the Google Client ID should be the client Id of the Oauth client id you generated. Your app Id should be the number before the “-“ of the client Id, and the secret can be found when you click on the Oauth client Id in your google cloud platform.



Google Service Account Email

googleiap@even-boulder-297520.iam.gserviceaccount.com

Google Package Name

com.bitHeads.GoogleIAP

Google App ID

[redacted]

Google Client ID

[redacted]

Google Client Secret

[redacted]

Google Service Account p12 Certificate

Select a certificate file

✓ Configured

That should be it for app setup!

For Verifying with braincloud you will need to authenticate, make a google purchase, and use our AppStore service to call VerifyPurchase.

In order to make this call you will need crucial information about your purchase, including the productId, orderId, and token of the purchase. On other platforms, a developer payload may also be needed, but in unity's case…

```
Developer payload is not supported
Google Play deprecated developer payload and is replacing it with alternatives that
are more meaningful and contextual.
```

In order to get this information we recommend you look at our braincloud example. In the BrainCloudInterface.cs we included a snippet to the regular purchase script logic.

Inside of Process Purchase we extract the info we need and store the values to be more easily accessible.

```
public    PurchaseProcessingResult    ProcessPurchase(PurchaseEventArgs    args)
    {
                        //    A    consumable    product    has    been    purchased
by    this    user.
                        if    (String.Equals(args.purchasedProduct.definition.id,
kProductIDConsumable,    StringComparison.Ordinal))
                        {
```

```
                                    //Debug.Log(string.Format("ProcessPurchase: 
PASS.    Product:    '{0}'",    args.purchasedProduct.definition.id));
                                    statusText    =    "ProcessPurchase:    PASS. 
Product:    "    +    args.purchasedProduct.definition.id;
                    }
                    else
                    {
                                    //Debug.Log(string.Format("ProcessPurchase: 
FAIL.    Unrecognized    product:    '{0}'", 
args.purchasedProduct.definition.id));
                                    statusText    =    "ProcessPurchase:    FAIL. 
Unrecognized    product:    "    +    args.purchasedProduct.definition.id;
                    }

                    wrapper    =    (Dictionary<string, 
object>)MiniJson.JsonDecode(args.purchasedProduct.receipt);
                    store    =    (string)wrapper["Store"];
                    payload    =    (string)wrapper["Payload"];
                    gpDetails    =    (Dictionary<string, 
object>)MiniJson.JsonDecode(payload);
                    gpJson    =    (string)gpDetails["json"];
                    gpSig    =    (string)gpDetails["signature"];

                    //    Return    a    flag    indicating    whether    this 
product    has    completely    been    received,    or    if    the    application 
needs
                    //    to    be    reminded    of    this    purchase    at 
next    app    launch.    Use    PurchaseProcessingResult.Pending    when    still
                    //    saving    purchased    products    to    the    cloud, 
and    when    that    save    is    delayed.
                    return    PurchaseProcessingResult.Complete;
    }
```

Then before we make our verify call, we need to create a String in a JSON format to pass up to the server. For simplicity to follow, we create a Dictionary to mimmick the JSON structure, then serialize it into a string that we pass into the VerifyPurchase call.

```
gpJsonDict    =    (Dictionary<string,    object>)MiniJson.JsonDecode(gpJson);

Dictionary<string,    object>    receiptData    =    new    Dictionary<string, 
object>();
receiptData.Add("productId",    gpJsonDict["productId"]);
receiptData.Add("orderId",    gpJsonDict["orderId"]);
receiptData.Add("token",    gpJsonDict["purchaseToken"]);
//Developer    payload    is    not    supported
//Google    Play    deprecated    developer    payload    and    is    replacing 
it    with    alternatives    that    are    more    meaningful    and 
contextual.
receiptData.Add("developerPayload",    "");    //So    pass    in    empty    string 
for    developer    payload.

string    receiptDataString    =    JsonWriter.Serialize(receiptData);
```

```
BCConfig._bc.AppStoreService.VerifyPurchase("googlePlay",    receiptDataString,
OnSuccess_VerifyPurchase,    OnError_VerifyPurchase);
```

Final Notes:

You will need to set up testers for your release track. In your Google Play Console, go to Testing > Closed Testing, where you likely made your Alpha release track, manage your track and add some testers. When sending the URL, we find that the "Join on the Web" link has success since we use a Web Application Oauth Client.

You should now be able to make a successful Google IAP and verify it with brainCloud. Note that testing in the editor will not work, and the Google Purchasing can only be done successfully and tested on an Android device.

Other useful links:

https://docs.unity3d.com/Manual/UnityIAPValidatingReceipts.html - Validating Recei

https://www.programmersought.com/article/87943674520/ - Purchase Script example

https://docs.revenuecat.com/docs/creating-play-service-credentials - Setting up API Access

https://developer.android.com/google/play/billing/test – Test Google Play Billing Integration

http://help.getbraincloud.com/en/articles/4736444-store-integration-google – Store Integration Info