Upgrading to brainCloud from PlayFab

Virtual Currency

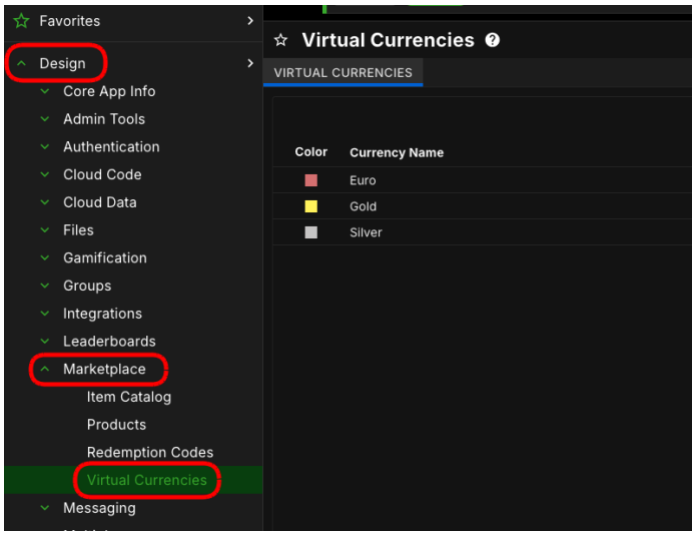
Table of Contents

Virtual Currency Introduction

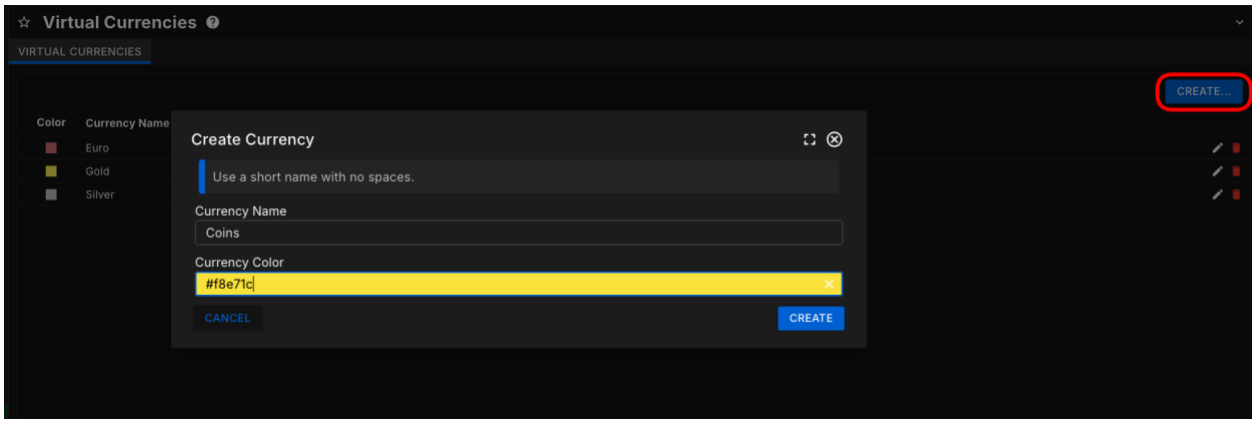
Virtual Currency (VC) is a standard feature for many backend game platforms and integrates into other marketplace and storefront features. Typically VC refers to soft-currency which is used in-game to purchase items without the need for in-app purchases from 3rd party stores like Google or Apple. brainCloud also provides VCs for use within its marketplace. In this section we will take a look at how VCs compare between brainCloud and PlayFab and how VC configuration from PlayFab can be replicated in brainCloud.

Configuration

Virtual Currencies in brainCloud are configurable from the **Design → Marketplace** menu.



You can click on the **Create** button to create a new VC. With brainCloud, you only have the ability to give your VC a name and a colour, with the colour mainly being used to differentiate your VCs in the portal.

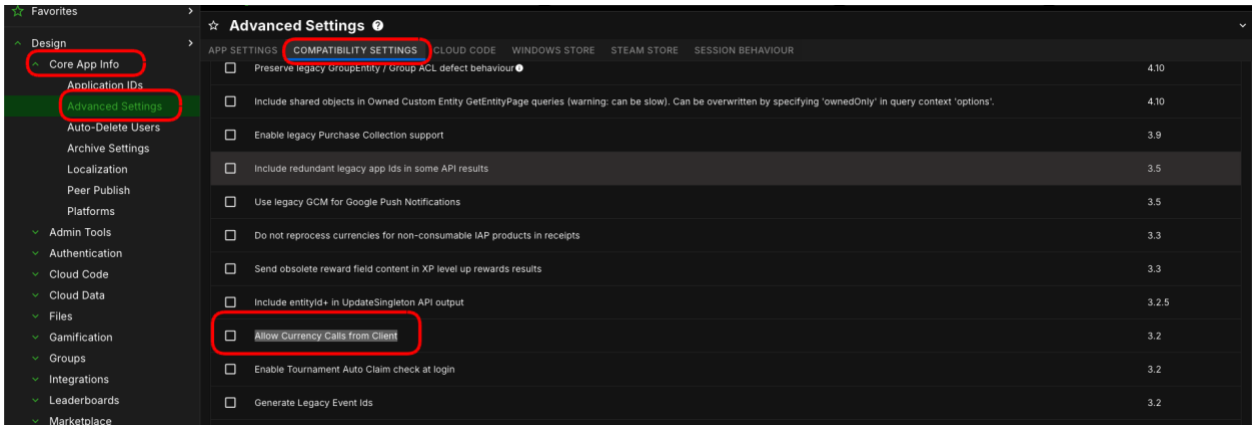


This means that brainCloud is missing some functionality for deposit and recharge options that PlayFab offers. We will explore how to add this functionality in a section below.

Fetching, Awarding and Debiting Virtual Currencies

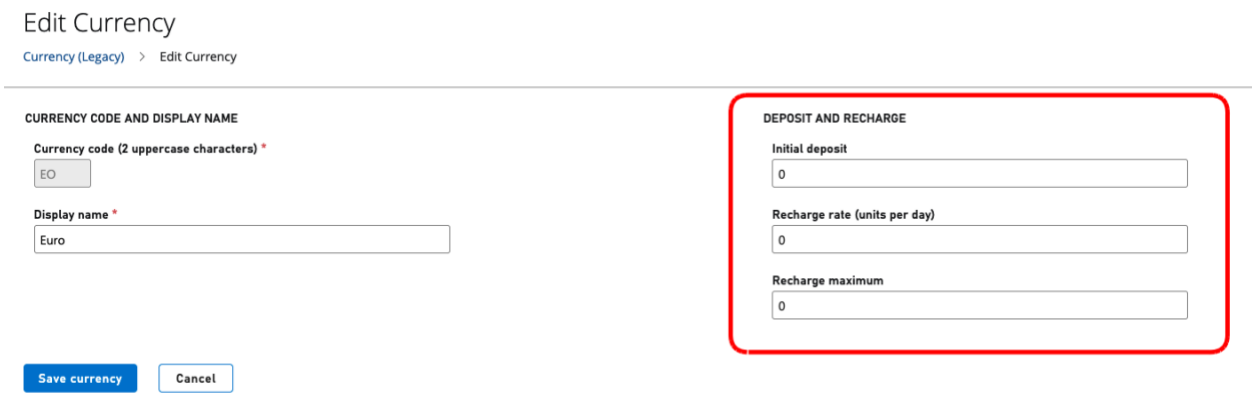
As with PlayFab, brainCloud offers a set of APIs for managing Virtual Currencies in your game allowing players to get balances, credit and debit currencies.

Something important to note is that the crediting of VCs from your game-client is disabled by default on brainCloud just as it is on PlayFab. This is done to protect your game's economy in the event that hackers or players misuse VC APIs. However, if required, you can enable this behaviour by going to Advanced Settings in the **Design → Core App** Info menu. Click on the **Compatibility Settings** tab and scroll down to find the **Allow Currency Calls from Client** option and enable it.



Deposit & Recharge

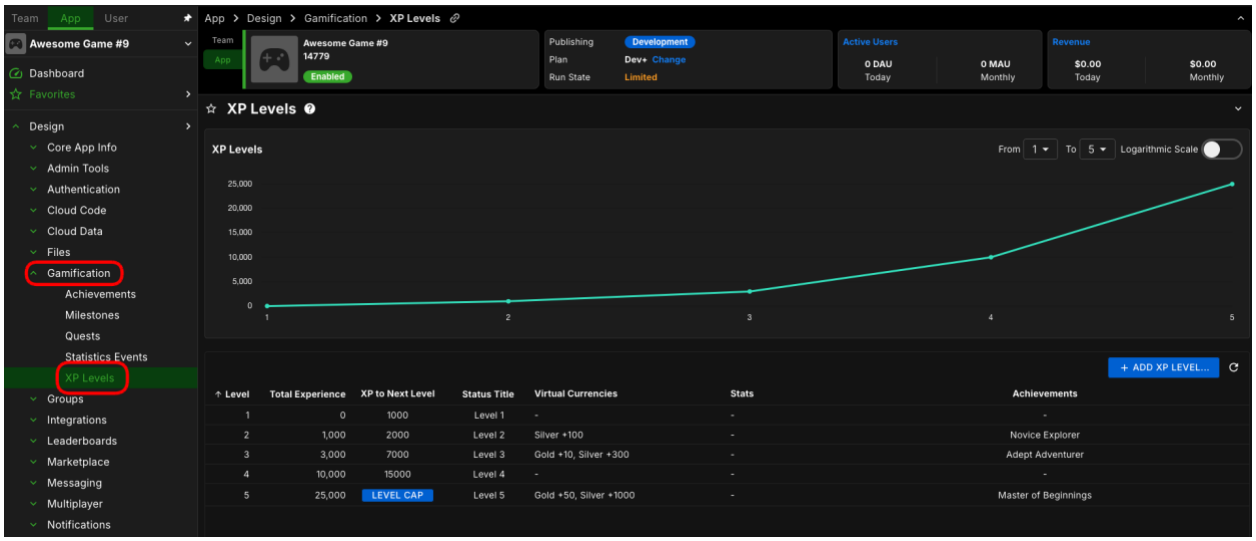
PlayFab allows developers to recharge VCs over the course of the day, and to have players start off with a given amount of currency when their account is created.



The most common feature from these options is the initial deposit, whereby the player is automatically delivered VC when their account is created. Let's take a look at how we could add this functionality with brainCloud using out-of-the-box features or by adding custom code.

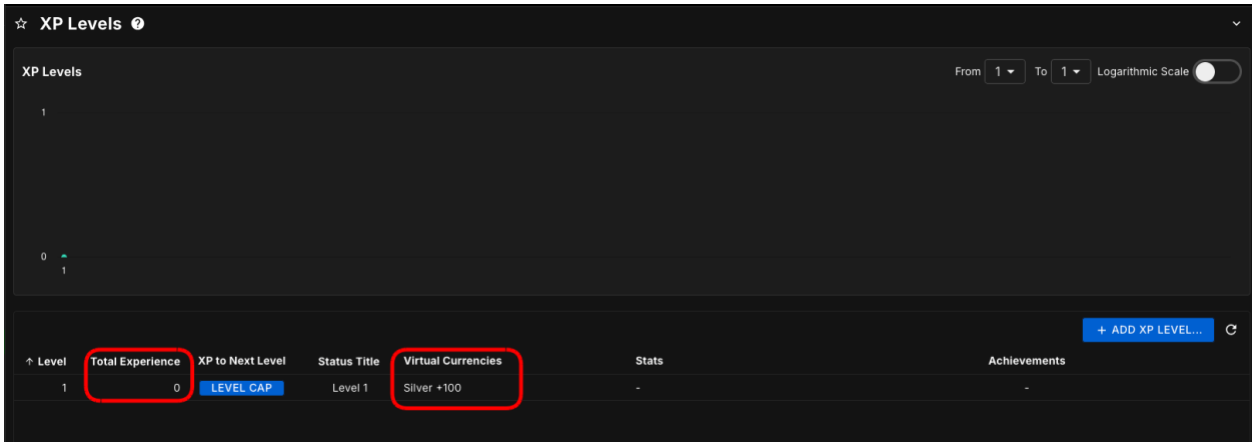
XP Levels

brainCloud offers a feature called XP Levels which allows developers to configure how players progress through levels based on their XP. This feature can also be used to deliver Virtual Goods, Achievements, and VCs when each level is reached. The XP Levels feature can be found in the **Design → Gamification** menu. Below is a simple example of how XP levels work in brainCloud.

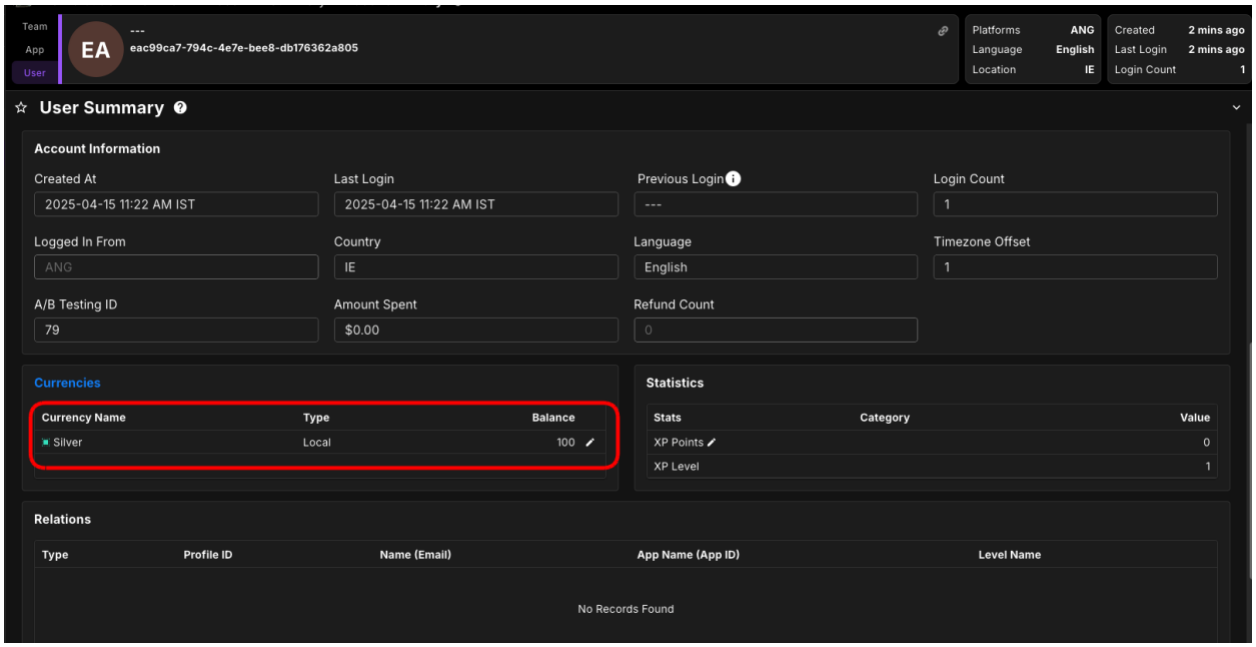


You can see from the list of levels, how much XP is required for the level to reach the next level and the VCs delivered.

While your game might not require XP or level progression, you can see in the example above that Level 1 requires 0 Total Experience. Therefore, all new players will trigger this level up as soon as their account is created. You can set your XP Levels up as in the example below to automatically deliver VC to the player when their account is created.



To test this, check out our section on the API Explorer in the Cloud Scripts document where you login and create new players from the brainCloud portal. You will see that any new player will be granted 100 silver the first time they log in.



Check out the section in the Player & Player Data Document for more information about the brainCloud User Browser, or brainCloud's docs here.

Custom Code Solution

Similar to PlayFab, brainCloud offers a custom cloud-code feature which can be used to execute custom functionality for a variety of purposes. You can also have code executed based on triggers, similar to PlayFab's Automation Rules.

For this example we want to have a set amount of currency delivered to the player after they log in, but only if this is a new player. Implementing this feature is actually already covered in another more comprehensive section on Cloud Scripts, so check that out for more information.

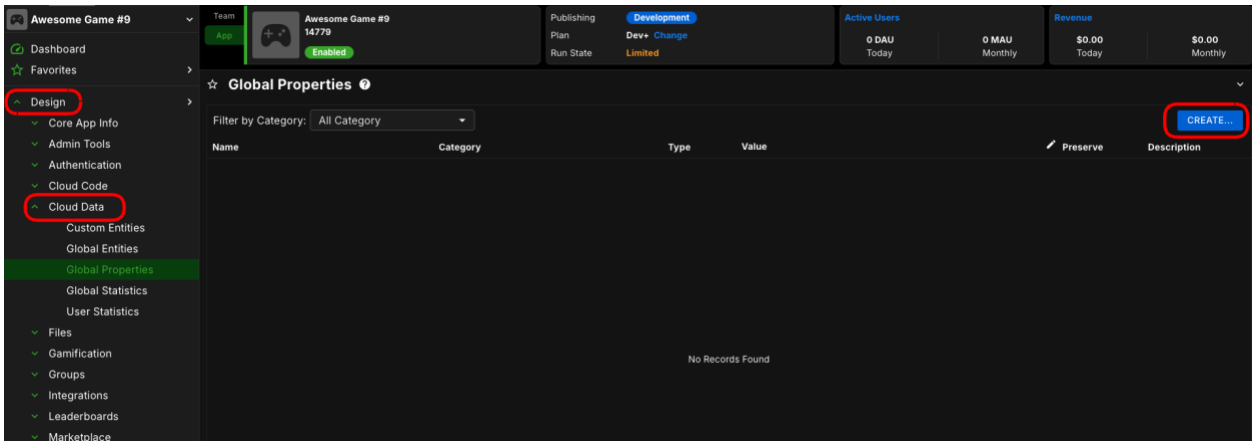
Recharging Virtual Currency

While brainCloud does not offer an out-of-the-bx feature for recharging VC, this can be achieved using custom code. Similar to the example mentioned above, this would be done using an API Hook and a custom script.

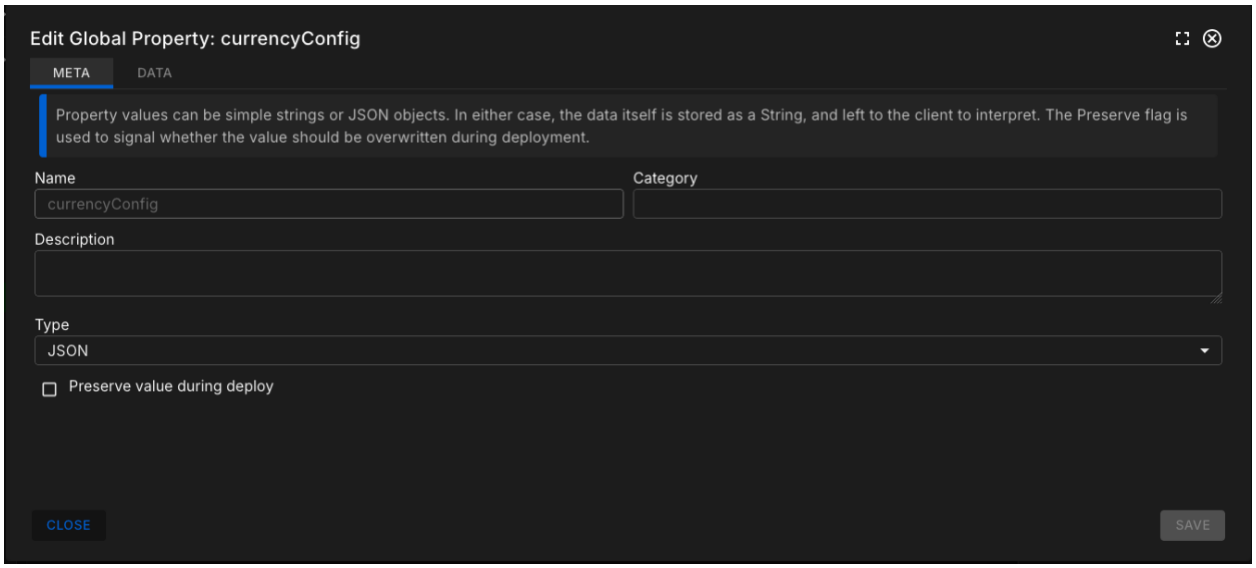
We already have a more thorough section on cloud-code, which also covers API Hooks, so we won't go into that here (check out the section on API Hooks in the Cloud Scripts Document for more details) but we can outline how to set this up in brief with some code examples.

Recharge Configuration

First we need a configuration file which will contain the details we need for how VCs are recharged. We can use Global Properties for this. **Global Properties** can be found in the **Design → Cloud Data** menu.

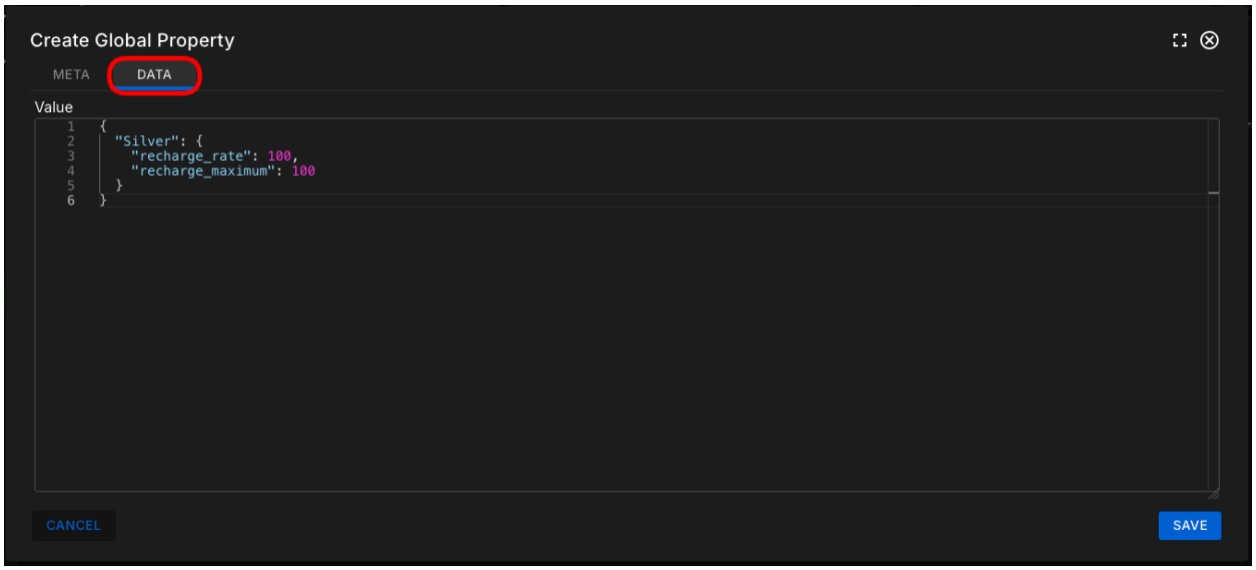


Create a new Global Property called “currencyConfig” and set its type to JSON. You can leave the rest of the details default.



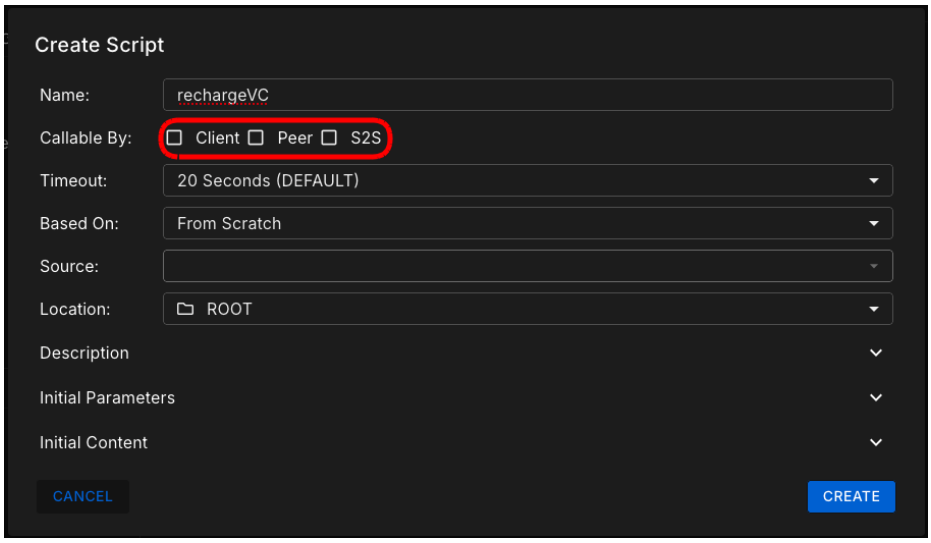
Click on the DATA tab and add the following JSON and click Save.

```
{
 "Silver": {
          "recharge_rate": 100,
          "recharge_maximum": 100
 }
}
```



The important thing here is that the name of your VC is correct. You can add more VC to this JSON for setting up recharge on additional VCs in your game.

Next, create a new script called “rechargeVC” and make sure that this script cannot be called by the client. You can leave the rest of the attributes as default.



This script is going to perform a number of actions, so let's breakdown what we need into steps:

1. **Load Global Properties**
   We'll need to check on the recharge rate and the max recharge for the day.

2. **Load User Attributes**
   We are going to store details about the recharge for this player in their User Attributes. This will allow us to load and update details whenever we add currency in order to track the date of the last recharge.

3. **Calculate The Recharge**
   We can do this by comparing the number of hours since the last recharge and then calculating the value of currency to deliver the player. PlayFab's recharge is continuous throughout a 24hr period, so we can divide the recharge amount by the number of hours since the last recharge in order to get the amount to be delivered to the player.

4. **Grant Virtual Currency**
   With the VC amount calculated we can grant the player the amount of VC they are entitled to.

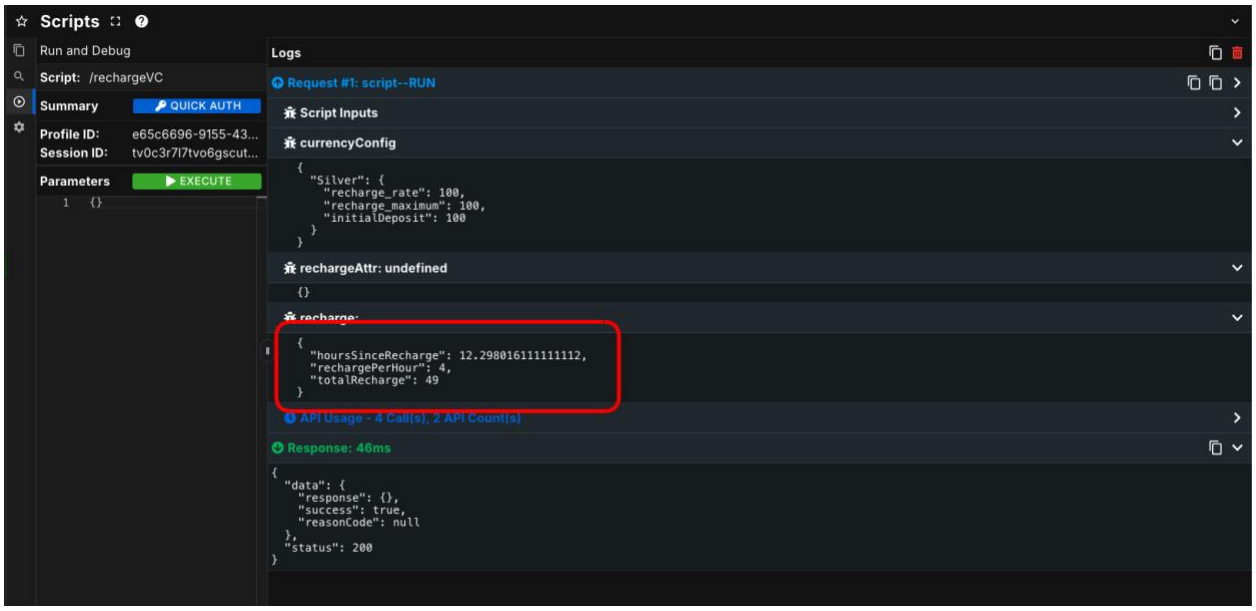
5. **Update Player Attribute**
   Lastly, we want to save back the timestamp for the last recharge for each currency so we can check them again later.

```
"use strict";


function main() {
 var response = {};


 bridge.logDebugJson("Script Inputs", data);


 // declare some constants we need for calculations //
 const now = new Date();
 const startOfDay = new Date(now.getFullYear(), now.getMonth(), now.getDate());
 const propertyName = "currencyConfig";
 const rechargeAttrName = "vcRechargeDetails";


 // [1] - Load Global Property //
 const globalAppProxy = bridge.getGlobalAppServiceProxy();
 const propResult = globalAppProxy.readSelectedProperties([propertyName]);
 if (propResult.status != 200) {
   // handler errors //
 }
 const currencyConfig = JSON.parse(propResult.data[propertyName].value);
 bridge.logDebugJson(propertyName, currencyConfig);


 // [2] - Load User Entity //
 const playerStateProxy = bridge.getPlayerStateServiceProxy();
 const attrResult = playerStateProxy.getAttributes();
 if (attrResult.status != 200) {
   // handler errors //
 }
 let rechargeAttr = attrResult.data.attributes[rechargeAttrName];
 bridge.logDebug("rechargeAttr: "+rechargeAttr);
```

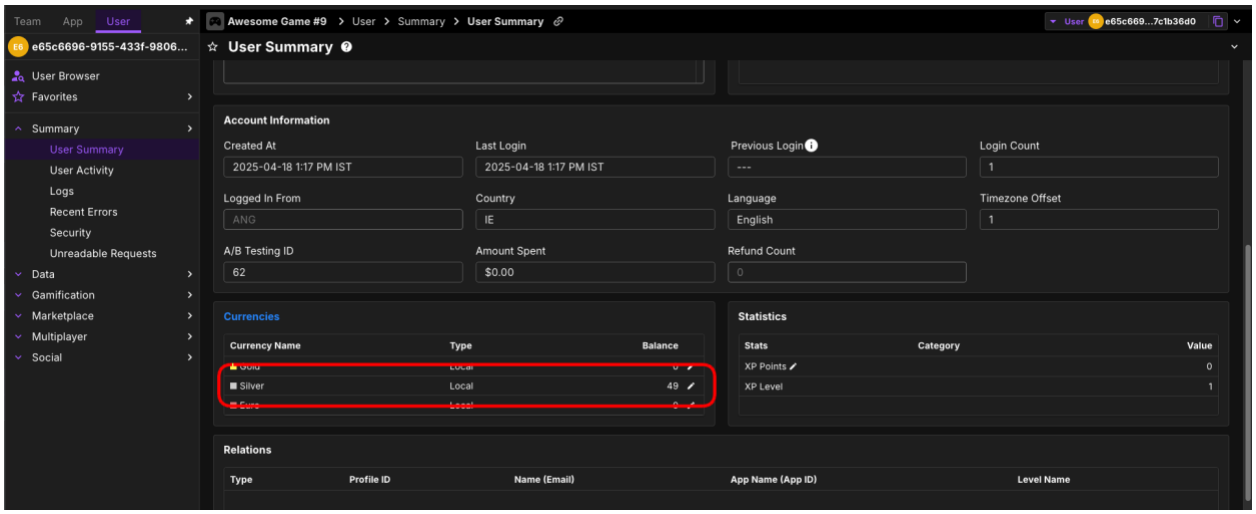
```
// Check attribute set up //

// If the player doesnt have the attribute, its their first time doing a recharge so we need to set it up //

if(rechargeAttr == undefined){

  rechargeAttr = {};

  Object.keys(currencyConfig).forEach(vc => {

    rechargeAttr[vc] = startOfDay.getTime();

  });

}

else{

  rechargeAttr = JSON.parse(rechargeAttr);

}


// [3] - For all currencies check how many hours since the last recharge //

const virtualCurrencyProxy = bridge.getVirtualCurrencyServiceProxy();

for(let vcKey in rechargeAttr){


  const hoursSinceRecharge = (now - new Date(rechargeAttr[vcKey])) / (1000 * 60 * 60);

  const rechargePerHour = Math.floor(currencyConfig[vcKey].recharge_rate/24);

  let totalRecharge = Math.floor(hoursSinceRecharge*rechargePerHour);


  bridge.logDebugJson("recharge: ", {

    hoursSinceRecharge: hoursSinceRecharge,

    rechargePerHour: rechargePerHour,

    totalRecharge: totalRecharge

  });


  // check for the max recharge //

  if(totalRecharge > currencyConfig[vcKey].recharge_maximum){

    totalRecharge = currencyConfig[vcKey].recharge_maximum;

  }
```

```
  if(totalRecharge > 0){
    // [4] - Deliver Currency //
    const awardVCResult = virtualCurrencyProxy.awardCurrency(vcKey, totalRecharge);
    if (awardVCResult.status != 200) {
      // handler errors //
    }
    // Update the last recharge date //
    rechargeAttr[vcKey] = Date.now();
  }
 }

 // [5] - Save Player Attribute //
 let attrUpdate = {};
 attrUpdate[rechargeAttrName] = JSON.stringify(rechargeAttr);
 const updateAttrResult = playerStateProxy.updateAttributes(attrUpdate, false);
 if (updateAttrResult.status != 200) {
   // handler errors //
 }
  return response;
}

main();
```

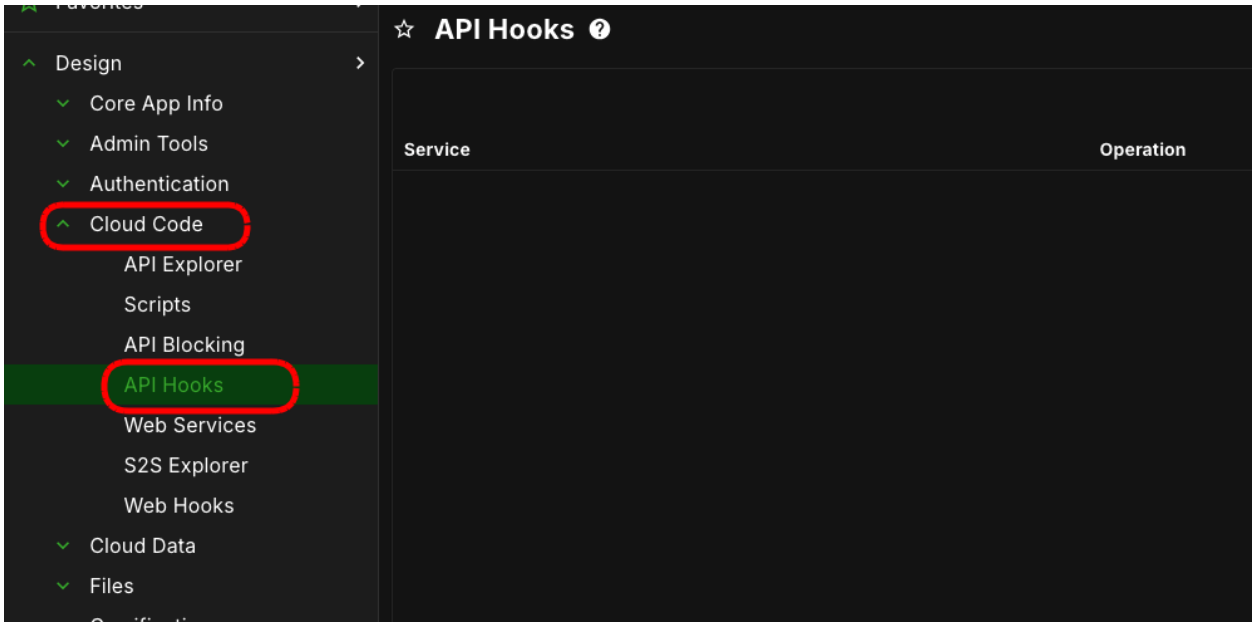
Running this code using the script debugger (see the section in the Cloud Scripts Document where we explore this feature further) we can test what happens when this script is executed.



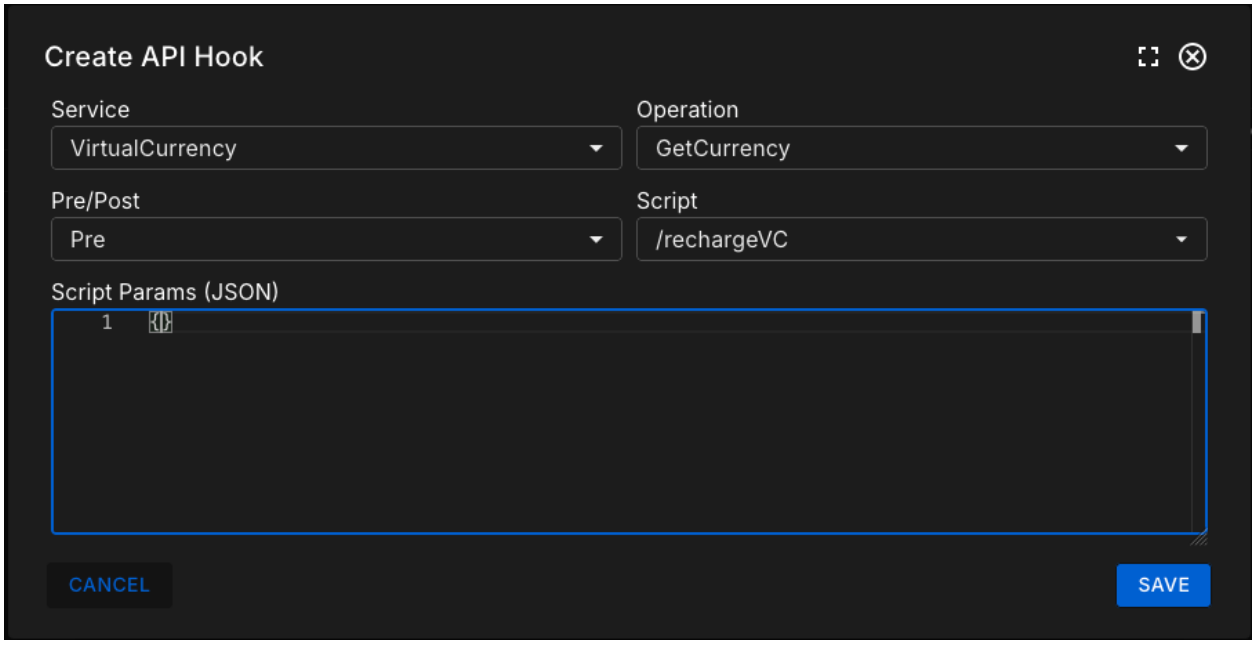
You can also double-check the VC actually got delivered by going to the User Profile for the player you are testing with.



The last step is to have this script execute automatically whenever the player gets their VC from the client. We can do this using API Hooks. You can find API Hooks in the **Design → Cloud Code** menu.



We want to create a new API Hook and use the **Virtual Currency** service and the **GetCurrency** operation. Make this a pre-Hook and select the rechargeVC script from the drop-down menu.



Now we can go back to the API Explorer to test this hook out (See the Cloud Scripts Document further details about using the API Explorer). In the API Explorer select the **Virtual Currency** service and the **GetCurrency** operation.

You will see the logs from our pre-hook and, if there is any VC to recharge, you will see it shown in the balance of the player through the response.

Note

For this example we had the VC recharge when the player calls GetCurrency from the client. In order to keep the player's balance up to date there may be other places where it would be useful to trigger this, such as post-authentication or after purchasing an item.

Summary

In this section we explored how to set up and use Virtual Currency using brainCloud and how to make use of out-of-the-box and custom features in order to extend brainCloud to adapt to your PlayFab migration requirements.

Virtual Currencies are also applied to Items, Products and other features in brainCloud. To see how prices can be set on Items, or how VCs can be purchased through 3rd party storefronts, check out our tutorials in the Economy and Catalogue document.