Upgrading to brainCloud from PlayFab

Player Inventory

Table of Contents

Introduction 3
Granting Items 3
Getting/Checking Item Details 4
Buying Items 5
Consuming & Revoking Items 5
Trading Items 6
  Open Trade 6
  GetTradeStatus 12
  Cancelling Trades 14
  Accepting Trades 15
Inventory API Comparison 20

Introduction

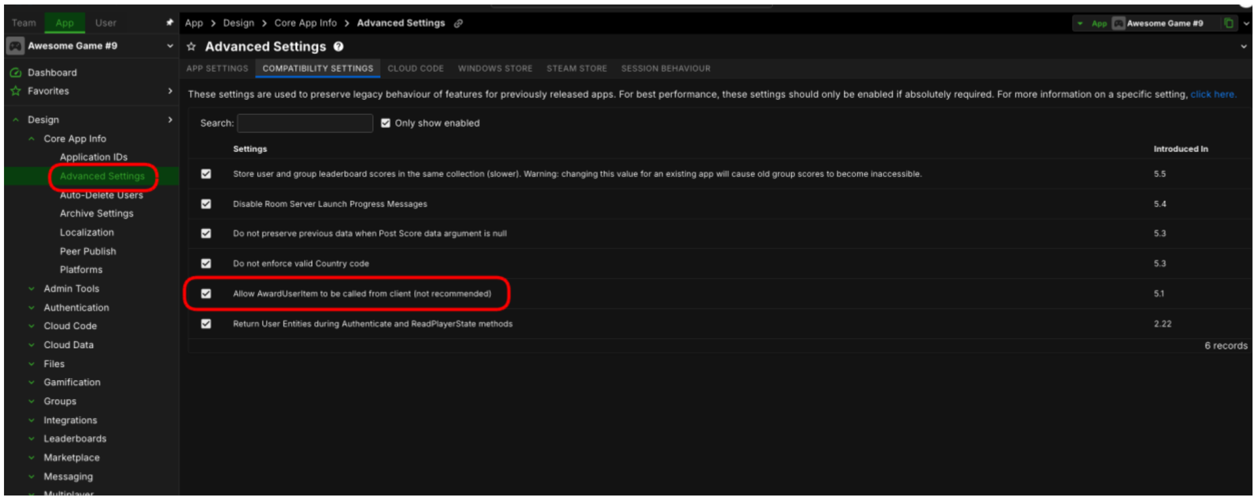
In the previous guide Economy and Catalogues, we explored how economy features work in brainCloud and the setup of the Item Catalog and Cash Product Store Offers. This guide will continue from there, covering general examples of player inventory management, such as granting/rewarding items and trading.

Granting Items

Some basic examples of granting/awarding items were covered in the Economy guide, but we'll revisit these requests with more detail here.

Outside purchasing from a third-party store, you can award items to a player using the AwardUserItem request. This is similar to PlayFab's **GrantItemsToUser**, but it works with one user and one item at a time.

It's crucial to note that, by default, you typically don't want to allow items to be awarded from the client, so brainCloud disables this feature. Instead, use a custom cloud-code script to validate the item before awarding it, preventing cheating. If you wish to bypass this protection, you *can* enable/disable client access to **AwardUserItem** from the **Core App Info → Advanced Settings** menu.

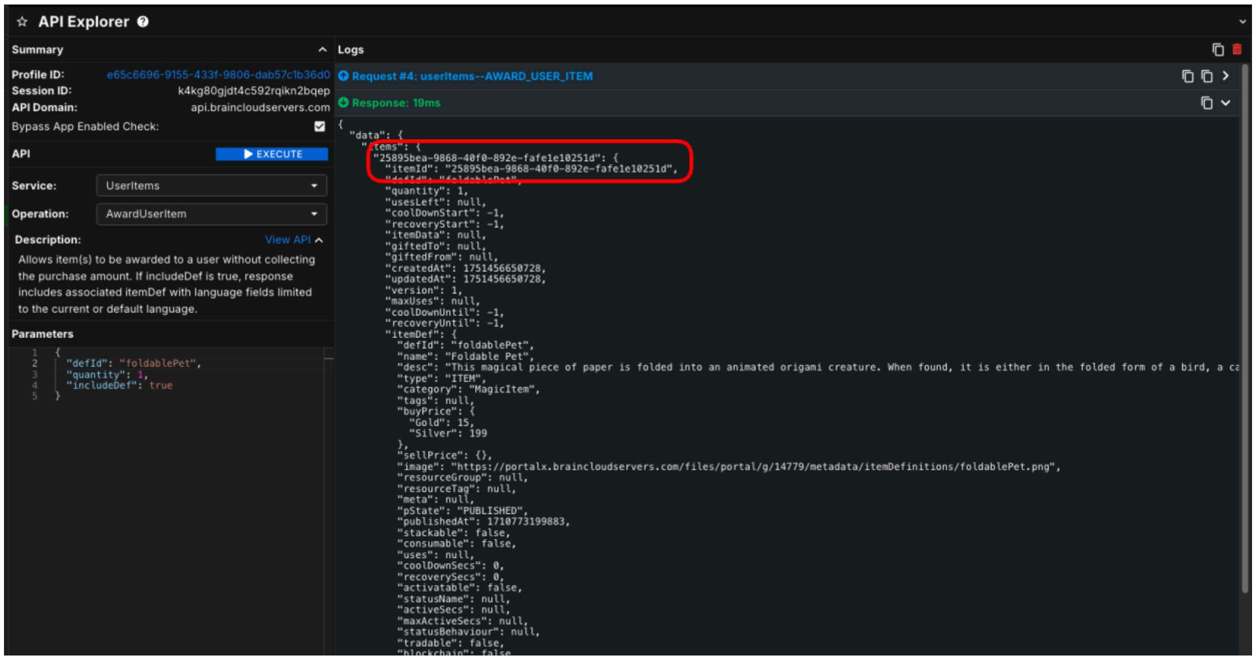


These settings are used to preserve legacy behaviour of features for previously released apps. For best performance, these settings should only be enabled if absolutely required. For more information on a specific setting, click here.

	Settings	Introduced In
✓	Store user and group leaderboard scores in the same collection (slower). Warning: changing this value for an existing app will cause old group scores to become inaccessible.	5.5
✓	Disable Room Server Launch Progress Messages	5.4
✓	Do not preserve previous data when Post Score data argument is null	5.3
✓	Do not enforce valid Country code	5.3
✓	Allow AwardUserItem to be called from client (not recommended)	5.1
✓	Return User Entities during Authenticate and ReadPlayerState methods	2.22

6 records

We can test this feature using the API Explorer. More information about the API Explorer is available in the Cloud Scripts Document. We will be using the **UserItems** service for this example.



Profile ID: e65c6696-9155-433f-9806-dab57c1b36d0
Session ID: k4kg80gjdt4c592rqikn2bqep
API Domain: api.braincloudservers.com
Bypass App Enabled Check: ✓

API
Service: UserItems
Operation: AwardUserItem
Description: Allows item(s) to be awarded to a user without collecting the purchase amount. If includeDef is true, response includes associated itemDef with language fields limited to the current or default language.

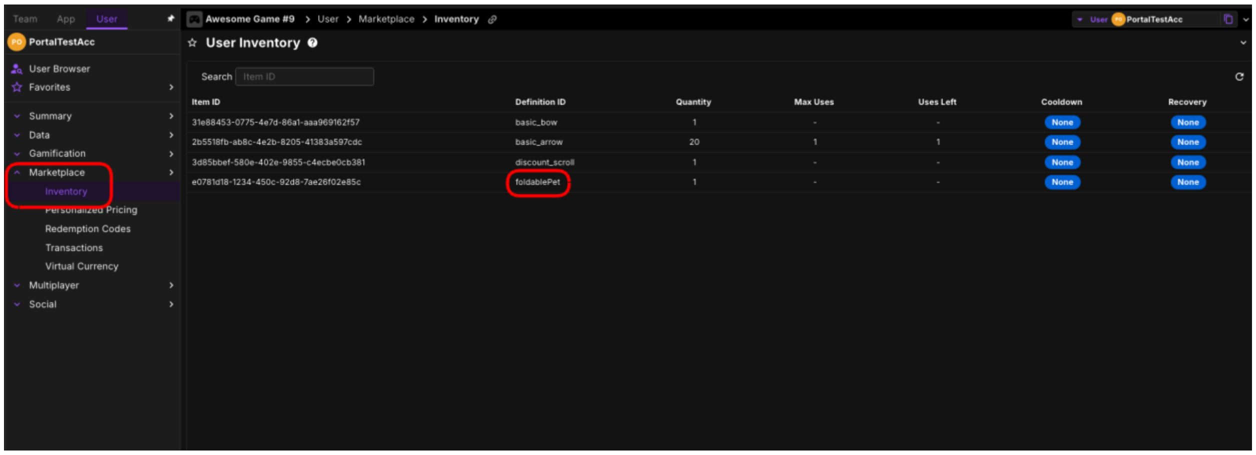
```
{
  "defId": "foldablePet",
  "quantity": 1,
  "includeDef": true
}
```

Request #4: userItems--AWARD_USER_ITEM
Response: 19ms

```
{
  "data": {
    "items": {
      "25895bea-9868-40f0-892e-fafe1e10251d": {
        "itemId": "25895bea-9868-40f0-892e-fafe1e10251d",
        "defId": "foldablePet",
        "quantity": 1,
        "usesLeft": null,
        "coolDownStart": -1,
        "recoveryStart": -1,
        "itemData": null,
        "giftedTo": null,
        "giftedFrom": null,
        "createdAt": 1751456650728,
        "updatedAt": 1751456650728,
        "version": 1,
        "maxUses": null,
        "coolDownUntil": -1,
        "recoveryUntil": -1,
        "itemDef": {
          "defId": "foldablePet",
          "name": "Foldable Pet",
          "desc": "This magical piece of paper is folded into an animated origami creature. When found, it is either in the folded form of a bird, a ca
          "type": "ITEM",
          "category": "MagicItem",
          "tags": null,
          "buyPrice": {
            "Gold": 15,
            "Silver": 199
          },
          "sellPrice": {},
          "image": "https://portalx.braincloudservers.com/files/portal/g/14779/metadata/itemDefinitions/foldablePet.png",
          "resourceGroup": null,
          "resourceTag": null,
          "meta": null,
          "pState": "PUBLISHED",
          "publishedAt": 1710773199883,
          "stackable": false,
          "consumable": false,
          "uses": null,
          "coolDownSecs": 0,
          "recoverySecs": 0,
          "activatable": false,
          "statusName": null,
          "activeSecs": null,
          "maxActiveSecs": null,
          "statusBehaviour": null,
          "tradable": false,
          "blockchain": false
```

You can see the response includes various information about the item. One of the most important attributes is the unique item ID. Keep note of this, as we'll use it later.

To verify delivery, go to the user's account dashboard and click on the **Marketplace → Inventory** menu.



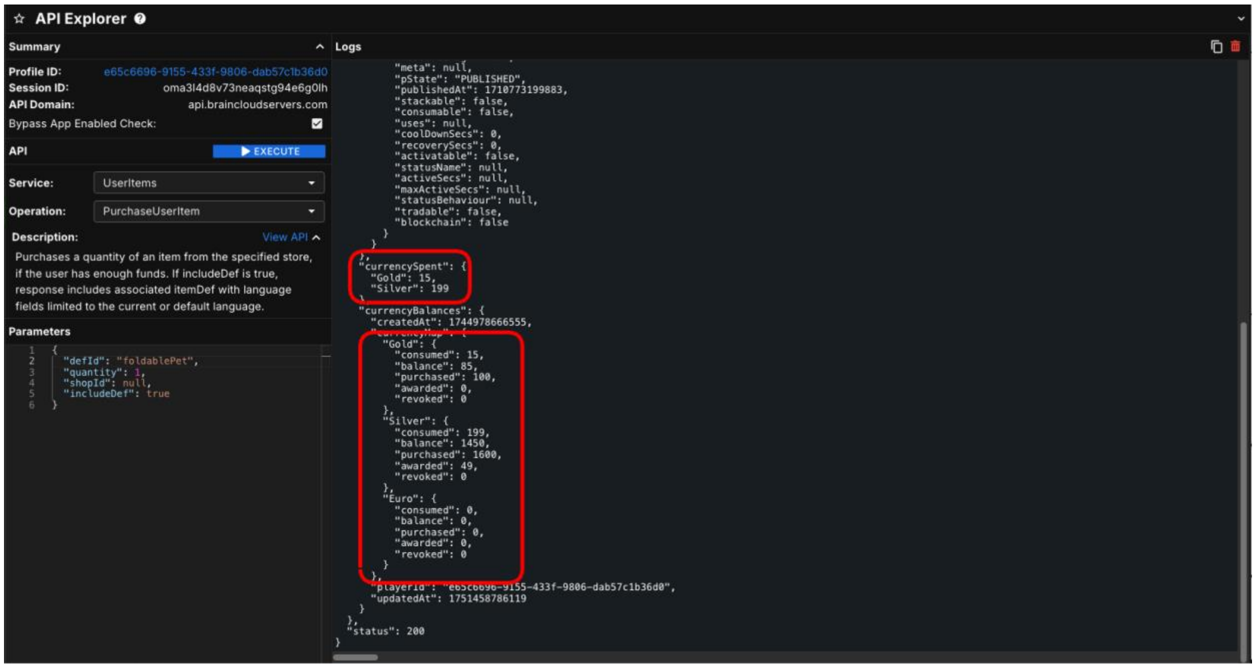
Item ID	Definition ID	Quantity	Max Uses	Uses Left	Cooldown	Recovery
31e88453-0775-4e7d-86a1-aaa969162f57	basic_bow	1	-	-	None	None
2b5518fb-ab8c-4e2b-8205-41383a597cdc	basic_arrow	20	1	1	None	None
3d85bbef-580e-402e-9855-c4ecbe0cb381	discount_scroll	1	-	-	None	None
e0781d18-1234-450c-92d8-7ae26f02e85c	foldablePet	1	-	-	None	None

Getting/Checking Item Details

PlayFab provides the **GetCharacterInventory** API, allowing players to receive a list of available inventory items. brainCloud offers two requests for this: GetUserItem retrieving details for a single item using its unique item ID, and GetUserItemsPage, returning a paginated, filterable list of items.

Buying Items

We've explored how to award an item for free to a user, but what if a player is purchasing from an ingame NPC store with virtual currency? Here, the player is granted the item, and their virtual currency is consumed instead of real-money. With PlayFab, this involves using both **GrantItemsToUser** and **SubtractUserVirtualCurrency**. In brainCloud, you can achieve this with the PurchaseUserItem request. The response from this request will include the currency consumed and a summary of the player's new balance.



Profile ID: e65c6696-9155-433f-9806-dab57c1b36d0
Session ID: oma3l4d8v73neaqstg94e6g0lh
API Domain: api.braincloudservers.com
Bypass App Enabled Check: ✓

Service: UserItems
Operation: PurchaseUserItem

Description:
Purchases a quantity of an item from the specified store, if the user has enough funds. If includeDef is true, response includes associated itemDef with language fields limited to the current or default language.

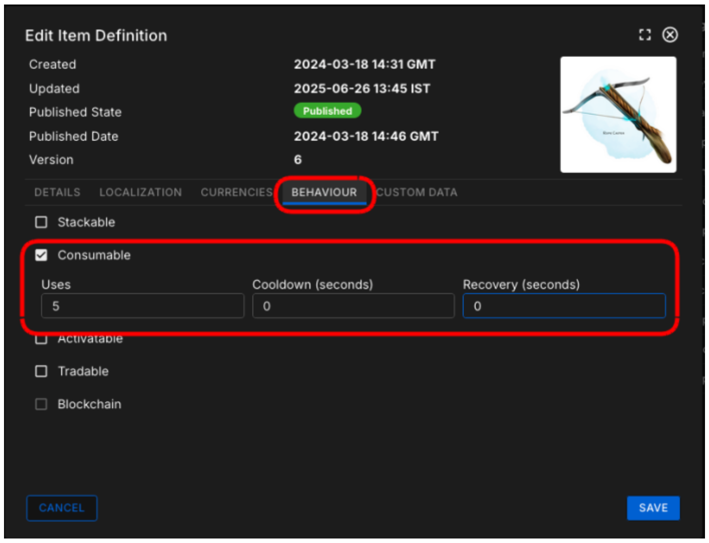
```
{
  "defId": "foldablePet",
  "quantity": 1,
  "shopId": null,
  "includeDef": true
}
```

```
        "meta": null,
        "pState": "PUBLISHED",
        "publishedAt": 1710773199883,
        "stackable": false,
        "consumable": false,
        "uses": null,
        "coolDownSecs": 0,
        "recoverySecs": 0,
        "activatable": false,
        "statusName": null,
        "activeSecs": null,
        "maxActiveSecs": null,
        "statusBehaviour": null,
        "tradable": false,
        "blockchain": false
      }
    }
  },
  "currencySpent": {
    "Gold": 15,
    "Silver": 199
  },
  "currencyBalances": {
    "createdAt": 1744978666555,
    "currencyMap": {
      "Gold": {
        "consumed": 15,
        "balance": 85,
        "purchased": 100,
        "awarded": 0,
        "revoked": 0
      },
      "Silver": {
        "consumed": 199,
        "balance": 1450,
        "purchased": 1600,
        "awarded": 49,
        "revoked": 0
      },
      "Euro": {
        "consumed": 0,
        "balance": 0,
        "purchased": 0,
        "awarded": 0,
        "revoked": 0
      }
    },
    "playerId": "e65c6696-9155-433f-9806-dab57c1b36d0",
    "updatedAt": 1751458786119
  }
},
"status": 200
}
```

Consuming & Revoking Items

You can remove items from a player's inventory using the DropUserItem request. This functions similarly to PlayFab's **RevokeInventoryItem**. However, if the item is stackable, **DropUserItem** will only remove one count from the stack until it reaches zero, acting also as a replacement for PlayFab's **ConsumeItem** call.

brainCloud provides another request related to consuming items: the UseUserItem request, which can be used to consume an item and remove it from the player's inventory, or, if configured to be consumable, reduce its "uses" by one.


Edit Item Definition

Created 2024-03-18 14:31 GMT
Updated 2025-06-26 13:45 IST
Published State Published
Published Date 2024-03-18 14:46 GMT
Version 6

DETAILS LOCALIZATION CURRENCIES BEHAVIOUR CUSTOM DATA

☐ Stackable
☑ Consumable
Uses 5
Cooldown (seconds) 0
Recovery (seconds) 0
☐ Activatable
☐ Tradable
☐ Blockchain

CANCEL SAVE

This flexible feature includes cooldowns and recovery periods for such items without needing custom cloud-code. You can read more about item behaviours here.

Trading Items

brainCloud offers APIs for gifting, selling, and receiving items from other players, which can replace the PlayFab trading API. However, to replace the specific trading flow PlayFab uses, custom cloud-code scripts are needed. We'll explore how to achieve that. Check out our guide here Cloud Scripts Document for more information on how to work with script before we continue.

Important

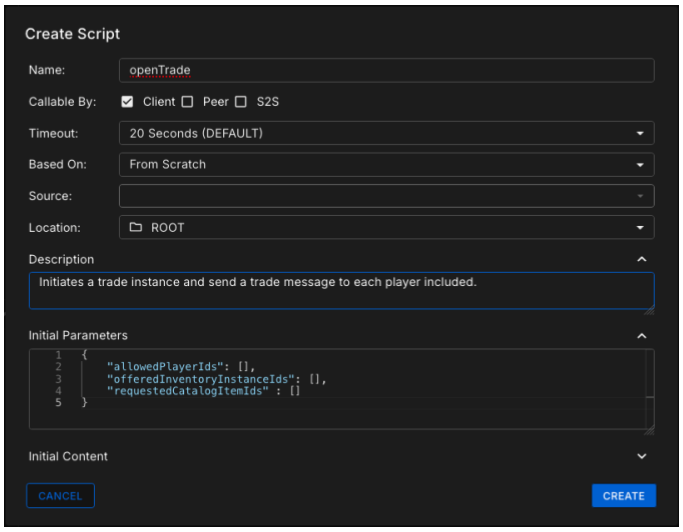
For this example, we'll use requests specific to trading. These require items to have their behavior set to "tradeable" for transfer between players.

Open Trade

The PlayFab trading flow starts with the **OpenTrade** API. This request sends a list of item instance IDs the player wishes to trade, a list of requested items, and the players they wish to invite to the trade.

Note
With PlayFab, if the requested item ID array is empty, the **OpenTrade** API will gift items to players who accept the trade. If this flow exists in your game, you can use the GiveItemToUser request instead of this custom example.

To begin, create a new custom cloud-code script called **openTrade**. Ensure this script is callable from the client.


Create Script

Name: openTrade

Callable By: [✓ Client] Peer S2S

Timeout: 20 Seconds (DEFAULT)

Based On: From Scratch

Source:

Location: ROOT

Description

Initiates a trade instance and send a trade message to each player included.

Initial Parameters

```
{
    "allowedPlayerIds": [],
    "offeredInventoryInstanceIds": [],
    "requestedCatalogItemIds" : []
}
```

Initial Content

Our script will perform the following actions:

1. Check if the player owns all the items they are offering.
2. Verify the requested items are valid (found in the item catalog).
3. Create an object for this trade log using custom entities.
4. Send trade details to all players.

Additionally, it would be useful to include item details in the response, so we'll add those while validating each item. Add the following code to your new script.

```
"use strict";


function main() {

 var response = {};
```

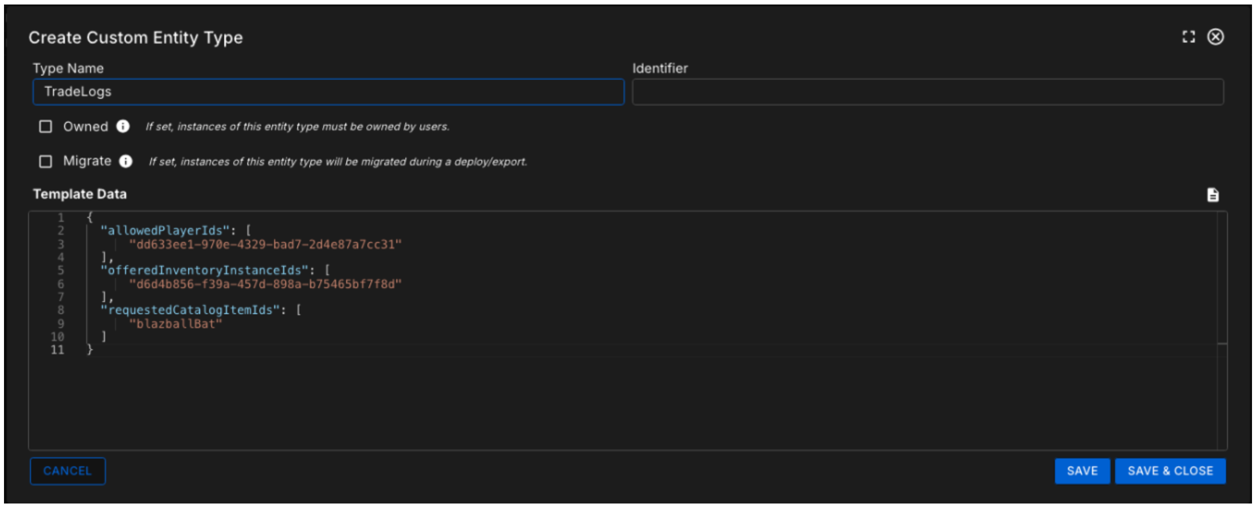
```
const allowedPlayerIds =            data.allowedPlayerIds;
let offeredInventoryInstanceIds =   data.offeredInventoryInstanceIds;
let requestedCatalogItemIds =       data.requestedCatalogItemIds;


// defId services //
const userItemsProxy =    bridge.getUserItemsServiceProxy();
const itemCatalogProxy =  bridge.getItemCatalogServiceProxy();
const customEntityProxy = bridge.getCustomEntityServiceProxy();
const messagingProxy =    bridge.getMessagingServiceProxy();


// trade log params //
const logTTL = 604800000; // 1 week in ms //
const isOwned = false;
const entityType = "TradeLogs";
const acl = { "other": 2 };


// [1] - Check the player owns the unique item //
offeredInventoryInstanceIds.forEach((uniqueId, index) => {
  let getUserItemResp = userItemsProxy.getUserItem(uniqueId, false);
  if (getUserItemResp.status != 200) {
    // your error here //
    // throw or return error;
  }
  offeredInventoryInstanceIds[index] = getUserItemResp.data.item;
});


// [2] - Check all the item IDs are valid //
requestedCatalogItemIds.forEach((defId, index) => {
  let itemDefResp = itemCatalogProxy.getCatalogItemDefinition(defId);
  if (itemDefResp.status != 200) {
    // your error here //
```

```
     // throw or return error;

   }

   requestedCatalogItemIds[index] = itemDefResp.data;

 });


 // [3] - Create new Trade Object //

let tradeLog = {

   requestedCatalogItemIds: requestedCatalogItemIds,

   offeredInventoryInstanceIds : offeredInventoryInstanceIds,

   allowedPlayerIds: allowedPlayerIds,

   status: "OFFERED",

   initiatedBy: bridge.getProfileId()

 };

 const createEntityResp = customEntityProxy.createEntity(entityType, tradeLog, acl,
logTTL, isOwned);

 if (createEntityResp.status != 200) {

   // your error here //

   // throw or return error;

 }

 tradeLog.tradeEntity = createEntityResp.data;


 // [4] - Send messages to each player with the trade log details //

 const sendMessResp = messagingProxy.sendMessage(allowedPlayerIds, tradeLog);

 if (sendMessResp.status != 200) {

   // your error here //

   // throw or return error;

 }


 response.tradeLog = tradeLog;

 return response;

}
```

```
main();
```

Note that in the example above, we've set a TTL for the trade log of one week. There's no reason for trades to remain longer, and we don't want the collection to store old or completed trades, so this helps clear them out.

Next, create a custom entity for the trade logs collection. To do this, go to the **Cloud Data → Custom Entities** menu and create a new custom entity collection.



Type Name: TradeLogs

Identifier:

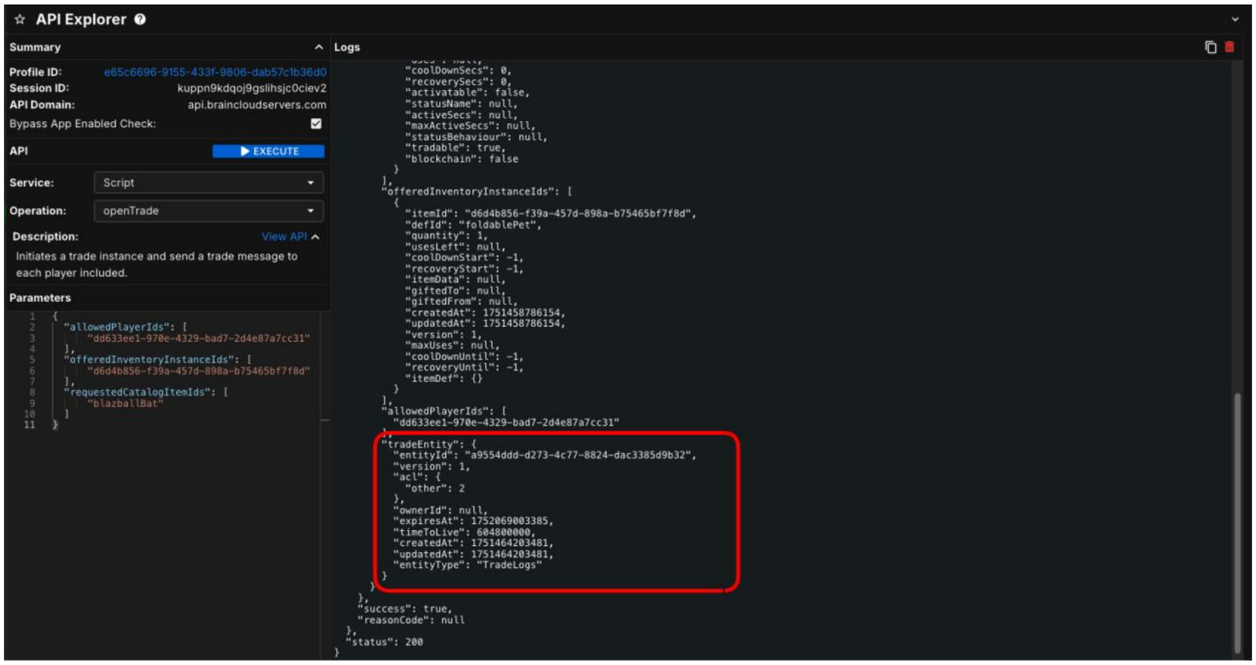
☐ Owned — *If set, instances of this entity type must be owned by users.*

☐ Migrate — *If set, instances of this entity type will be migrated during a deploy/export.*

Template Data

```
{
  "allowedPlayerIds": [
      "dd633ee1-970e-4329-bad7-2d4e87a7cc31"
  ],
  "offeredInventoryInstanceIds": [
      "d6d4b856-f39a-457d-898a-b75465bf7f8d"
  ],
  "requestedCatalogItemIds": [
      "blazballBat"
  ]
}
```

Now we can test this example using the API Explorer. You'll see in the response that we get the entity ID for the trade log. The players who received messages about the trade also get this ID and can use it to look up the trade in the future.



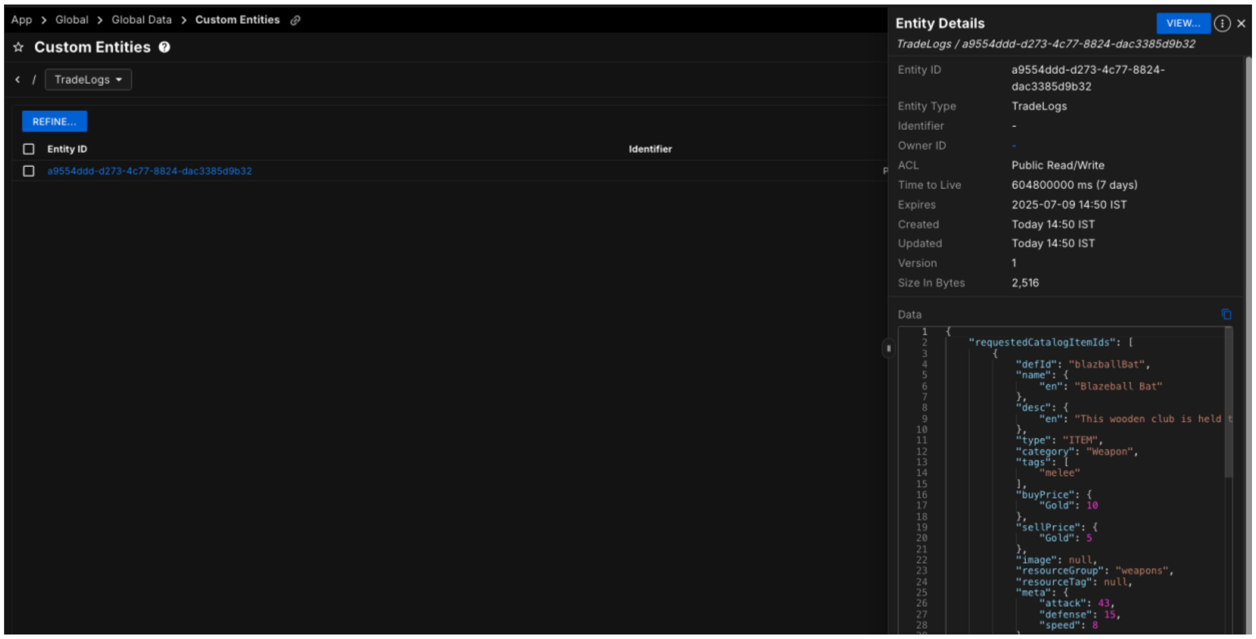
Profile ID: e65c6696-9155-433f-9806-dab57c1b36d0
Session ID: kuppn9kdqoj9gslihsjc0ciev2
API Domain: api.braincloudservers.com
Bypass App Enabled Check: ✓

Service: Script
Operation: openTrade
Description: Initiates a trade instance and send a trade message to each player included.

```
{
  "allowedPlayerIds": [
      "dd633ee1-970e-4329-bad7-2d4e87a7cc31"
  ],
  "offeredInventoryInstanceIds": [
      "d6d4b856-f39a-457d-898a-b75465bf7f8d"
  ],
  "requestedCatalogItemIds": [
      "blazballBat"
  ]
}
```

```
          "coolDownSecs": 0,
          "recoverySecs": 0,
          "activatable": false,
          "statusName": null,
          "activeSecs": null,
          "maxActiveSecs": null,
          "statusBehaviour": null,
          "tradable": true,
          "blockchain": false
        }
      ],
      "offeredInventoryInstanceIds": [
        {
          "itemId": "d6d4b856-f39a-457d-898a-b75465bf7f8d",
          "defId": "foldablePet",
          "quantity": 1,
          "usesLeft": null,
          "coolDownStart": -1,
          "recoveryStart": -1,
          "itemData": null,
          "giftedTo": null,
          "giftedFrom": null,
          "createdAt": 1751458786154,
          "updatedAt": 1751458786154,
          "version": 1,
          "maxUses": null,
          "coolDownUntil": -1,
          "recoveryUntil": -1,
          "itemDef": {}
        }
      ],
      "allowedPlayerIds": [
        "dd633ee1-970e-4329-bad7-2d4e87a7cc31"
      ],
      "tradeEntity": {
        "entityId": "a9554ddd-d273-4c77-8824-dac3385d9b32",
        "version": 1,
        "acl": {
          "other": 2
        },
        "ownerId": null,
        "expiresAt": 1752069003385,
        "timeToLive": 604800000,
        "createdAt": 1751464203481,
        "updatedAt": 1751464203481,
        "entityType": "TradeLogs"
      }
    }
  },
  "success": true,
  "reasonCode": null
},
"status": 200
}
```

You can confirm that the trade log was stored by going to the entity collection and clicking on the document. You'll see a summary of the log in the right-hand side panel.



Entity ID	Identifier
a9554ddd-d273-4c77-8824-dac3385d9b32	

TradeLogs / a9554ddd-d273-4c77-8824-dac3385d9b32

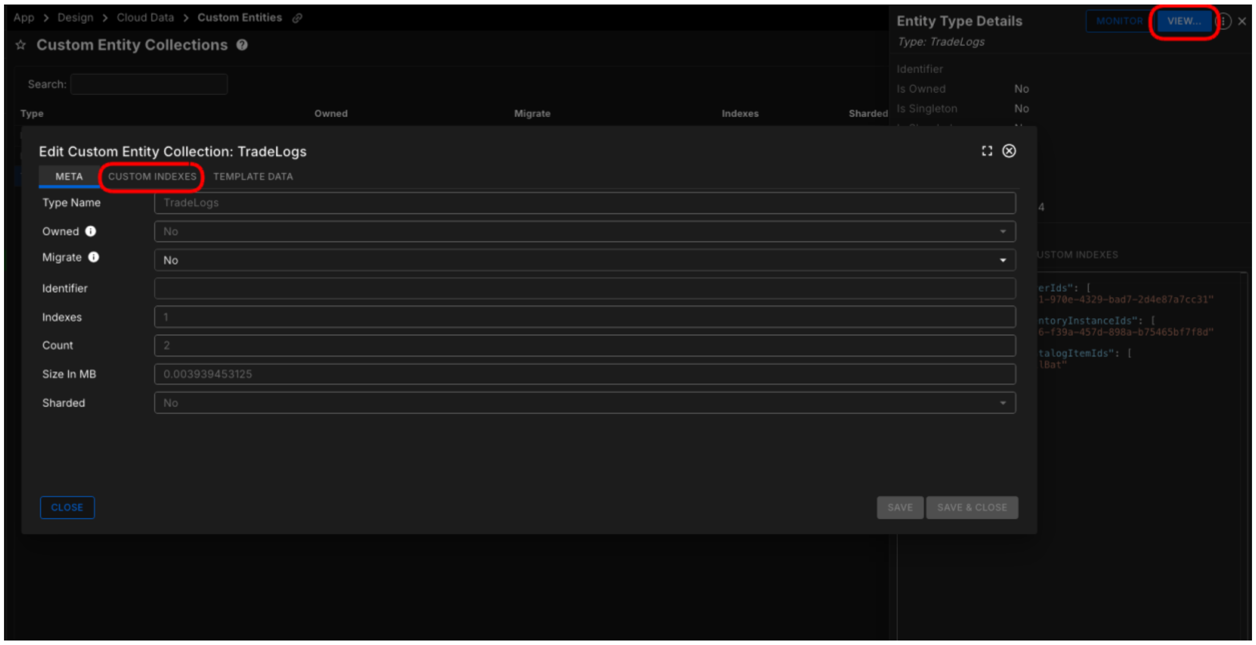
Entity ID	a9554ddd-d273-4c77-8824-dac3385d9b32
Entity Type	TradeLogs
Identifier	-
Owner ID	-
ACL	Public Read/Write
Time to Live	604800000 ms (7 days)
Expires	2025-07-09 14:50 IST
Created	Today 14:50 IST
Updated	Today 14:50 IST
Version	1
Size In Bytes	2,516

```
{
    "requestedCatalogItemIds": [
        {
            "defId": "blazballBat",
            "name": {
                "en": "Blazeball Bat"
            },
            "desc": {
                "en": "This wooden club is held t
            },
            "type": "ITEM",
            "category": "Weapon",
            "tags": [
                "melee"
            ],
            "buyPrice": {
                "Gold": 10
            },
            "sellPrice": {
                "Gold": 5
            },
            "image": null,
            "resourceGroup": "weapons",
            "resourceTag": null,
            "meta": {
                "attack": 43,
                "defense": 15,
                "speed": 8
```

GetTradeStatus

Players can view the details of this trade anytime using the ReadEntity. However, with PlayFab, there's also the option to view all of a player's trades using the **GetPlayerTrades** API. Let's see how we can modify our entity collection to allow this.

Go to your custom entity definition and click on the **View** button in the right-hand panel.



Type: TradeLogs

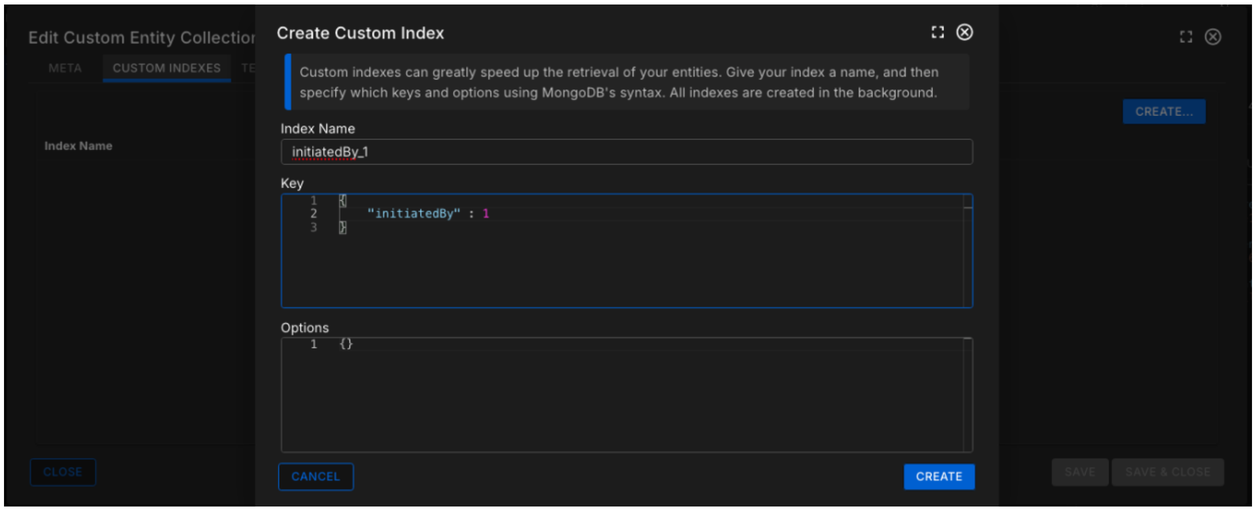
Identifier	
Is Owned	No
Is Singleton	No

Edit Custom Entity Collection: TradeLogs

Type Name	TradeLogs
Owned	No
Migrate	No
Identifier	
Indexes	1
Count	2
Size In MB	0.003939453125
Sharded	No

```
erIds": [
1-970e-4329-bad7-2d4e87a7cc31"
ntoryInstanceIds": [
6-f39a-457d-898a-b75465bf7f8d"
talogItemIds": [
lBat"
```

Click on the Custom Indexes tab and create a new index as below.



Create Custom Index

Custom indexes can greatly speed up the retrieval of your entities. Give your index a name, and then specify which keys and options using MongoDB's syntax. All indexes are created in the background.

Index Name: initiatedBy_1

Key

```
{
    "initiatedBy" : 1
}
```

Options

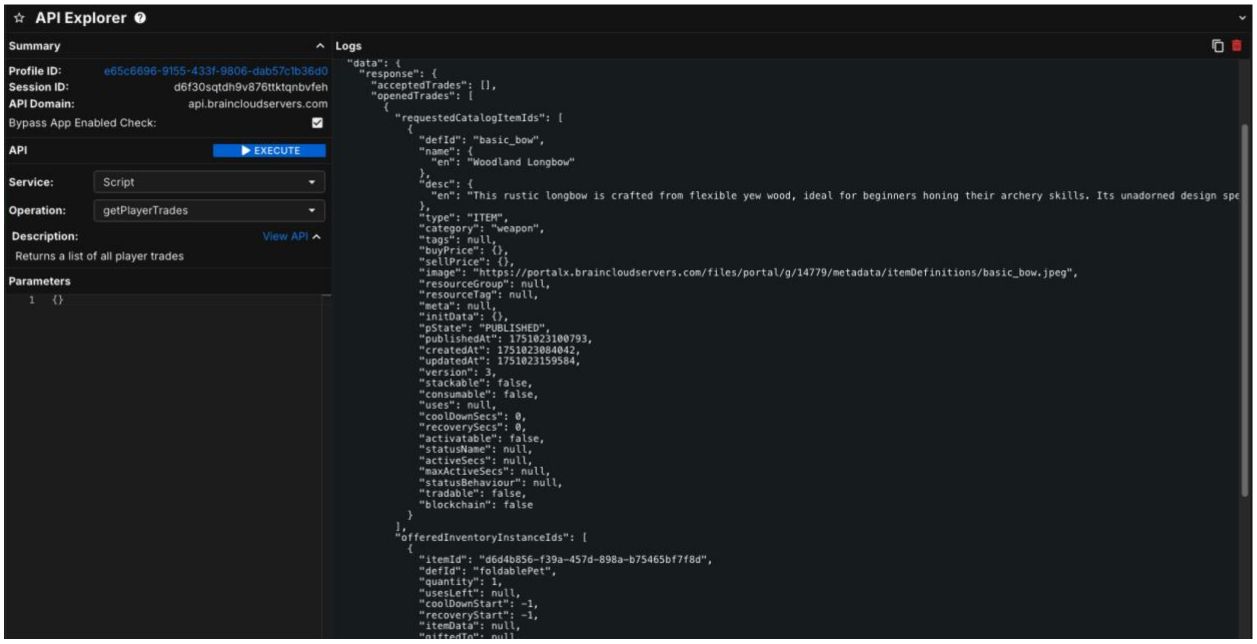
```
{}
```

Next, create a new custom cloud-code script called **getPlayerTrades**. Ensure this script is callable from the client, and add the following code to your new script.

```
"use strict";


function main() {

 var response = {};


 let acceptedTrades = [];

 let openedTrades = [];

 const playerId = bridge.getProfileId();


 // [1] - define query params //

 const entityType = "TradeLogs";

 const context = {

   "searchCriteria": {

     "data.initiatedBy": playerId

   }

 };


 // [2] - query entity collection //

 const customEntityProxy = bridge.getCustomEntityServiceProxy();

 const getTradesResp = customEntityProxy.getEntityPage(entityType, context);

 if (getTradesResp.status != 200) {

   // your error here //

   // throw or return error;

 }


 // [3] - filter trades into each array //

 getTradesResp.data.results.items.forEach(doc => {

   if(doc.data.status == "ACCEPTED"){

     acceptedTrades.push(doc.data);
```

```
    }
    else {
      openedTrades.push(doc.data);
    }
  });

  // [4] - Return arrays //
  response.acceptedTrades = acceptedTrades;
  response.openedTrades = openedTrades;
  return response;
}

main();
```

Testing this new script out from the API Explorer we can see that it returns our new open trade for this player.



Profile ID: e65c6696-9155-433f-9806-dab57c1b36d0
Session ID: d6f30sqtdh9v876ttktqnbvfeh
API Domain: api.braincloudservers.com
Bypass App Enabled Check: ✓
Service: Script
Operation: getPlayerTrades
Description:
Returns a list of all player trades

```
{}
```

```
"data": {
  "response": {
    "acceptedTrades": [],
    "openedTrades": [
      {
        "requestedCatalogItemIds": [
          {
            "defId": "basic_bow",
            "name": {
              "en": "Woodland Longbow"
            },
            "desc": {
              "en": "This rustic longbow is crafted from flexible yew wood, ideal for beginners honing their archery skills. Its unadorned design spe
            },
            "type": "ITEM",
            "category": "weapon",
            "tags": null,
            "buyPrice": {},
            "sellPrice": {},
            "image": "https://portalx.braincloudservers.com/files/portal/g/14779/metadata/itemDefinitions/basic_bow.jpeg",
            "resourceGroup": null,
            "resourceTag": null,
            "meta": null,
            "initData": {},
            "pState": "PUBLISHED",
            "publishedAt": 1751023100793,
            "createdAt": 1751023084042,
            "updatedAt": 1751023159584,
            "version": 3,
            "stackable": false,
            "consumable": false,
            "uses": null,
            "coolDownSecs": 0,
            "recoverySecs": 0,
            "activatable": false,
            "statusName": null,
            "activeSecs": null,
            "maxActiveSecs": null,
            "statusBehaviour": null,
            "tradable": false,
            "blockchain": false
          }
        ],
        "offeredInventoryInstanceIds": [
          {
            "itemId": "d6d4b856-f39a-457d-898a-b75465bf7f8d",
            "defId": "foldablePet",
            "quantity": 1,
            "usesLeft": null,
            "coolDownStart": -1,
            "recoveryStart": -1,
            "itemData": null,
            "giftedTo": null
```

Cancelling Trades

Canceling trades is straightforward with the example above. To simply remove the trade, use the DeleteEntity request. However, if you want the trade log marked as "CANCELLED" but still readable by participants, you'll need to write a script for this. We'll explore updating a custom entity in the next example, so we'll skip the script for **CancelTrade** for now.

Accepting Trades

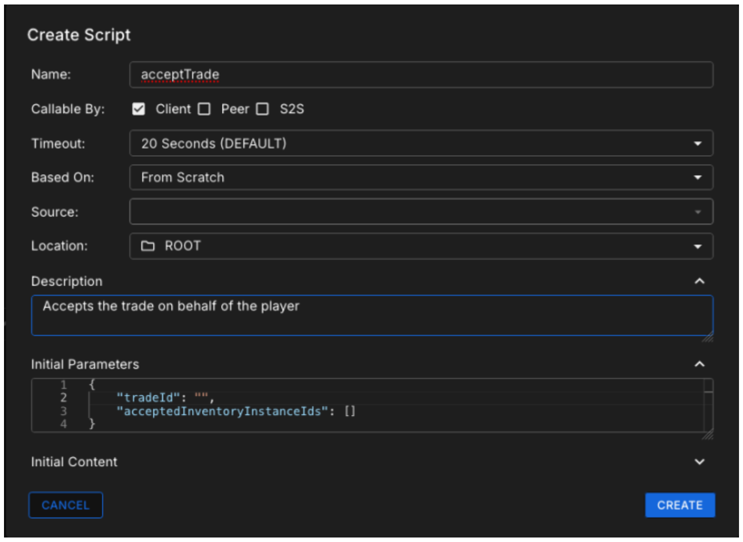
The final part of PlayFab's trading flow is the **AcceptTrade** API. This call swaps items between players and updates the trade log.

brainCloud offers some out-of-the-box requests which do not require custom scripts and might suit your use-case. For example, SellItemToUser could be used in a case where a player only wants to trade for virtual currency and GiveUserItem followed by ReceiveUserItemFrom can exchange items between players for free. However, to replicate PlayFab's functionality exactly, you'll need to create a custom script.

To begin, create a new custom cloud-code script called **acceptTrade**. Ensure that it's callable from the client. Our script will perform the following actions:

1. Load the trade log.
2. Check if the trade is still open.
3. Confirm the player accepting is included in the trade list.
4. Verify the player still has the items they want to exchange.
5. Deliver the items.
6. Update the trade log to mark it as accepted.
7. Send a trade update message to all players.

PlayFab's **AcceptTrade** API requires the player's unique item IDs, the trade ID, and the offering player ID. In this case, we only need the trade ID and the item IDs, as the other player's ID is already stored in the trade log.


Create Script

Name: acceptTrade

Callable By: [✓ Client] Peer S2S

Timeout: 20 Seconds (DEFAULT)

Based On: From Scratch

Source:

Location: ROOT

Description

Accepts the trade on behalf of the player

Initial Parameters

```
{
    "tradeId": "",
    "acceptedInventoryInstanceIds": []
}
```

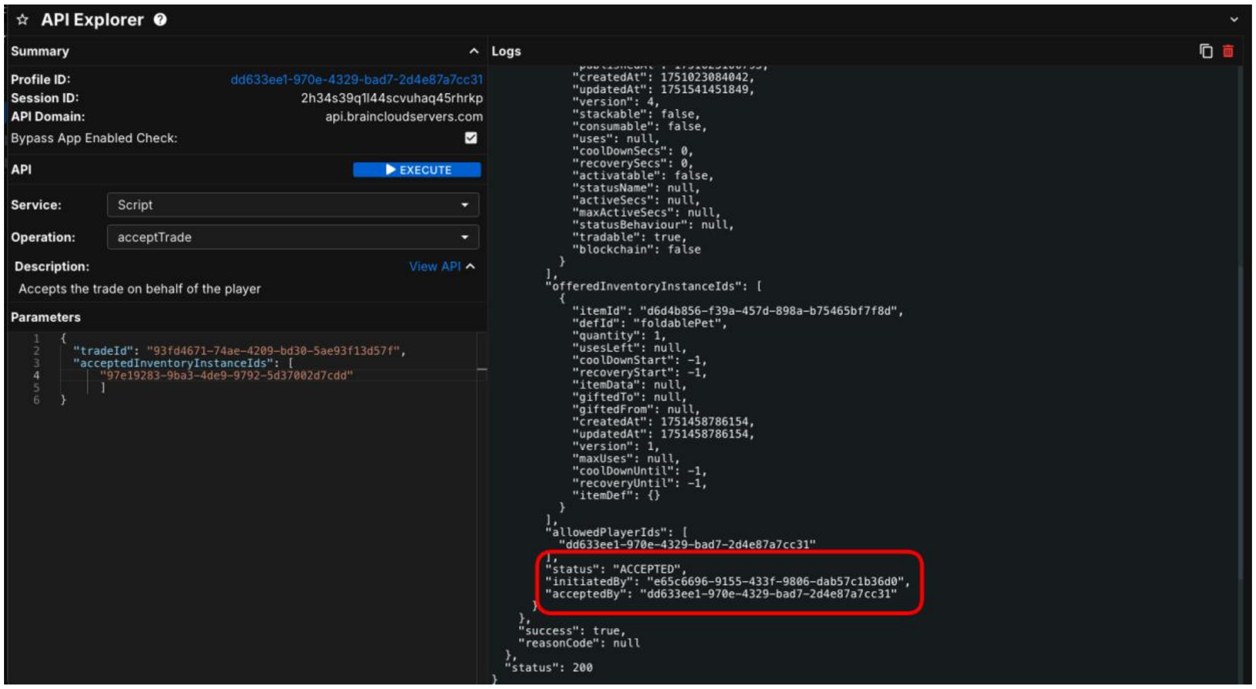
Initial Content

Add the following code to your new script.

```
"use strict";


function main() {

 var response = {};

 // params //

 const entityType =                      "TradeLogs";

 const logTTL =                          604800000; // 1 week in ms //

 const entityId =                        data.tradeId;

 const acceptedInventoryInstanceIds =    data.acceptedInventoryInstanceIds;

 let playerId =                          bridge.getProfileId();


 // define services //

 const customEntityProxy = bridge.getCustomEntityServiceProxy();

 const userItemsProxy =    bridge.getUserItemsServiceProxy();

 const messagingProxy =    bridge.getMessagingServiceProxy();


 // [1] - Load the trade Log //

 const readEntityResp = customEntityProxy.readEntity(entityType, entityId);

 if (readEntityResp.status != 200) {
   // your error here //

   // throw or return error;

 }

 let entity =    readEntityResp.data

 let tradeLog =  readEntityResp.data.data;


 // [2] - Check if the trade is still open //

 if(tradeLog.status == "ACCEPTED"){

   throw "your error here - trade is no longer open";

 }


 // [3] - Check if this player is in the list //

 if(tradeLog.allowedPlayerIds.includes(playerId)){

   throw "your error here - player not found in trade";

 }


 // [4] - Check if current player has the item(s) submitted for the trade //

 //         First check if player has actually submitted items //

 if(acceptedInventoryInstanceIds.length == 0){

   throw "your error here - acceptedInventoryInstanceIds empty";

 }

 const context = { "searchCriteria": { "itemId": { "$in" : acceptedInventoryInstanceIds
}}};

 const getPlayerItems = userItemsProxy.getUserItemsPage(context, true);

 if (getPlayerItems.status != 200) {

   // your error here //

   // throw or return error;

 }

 const playerItems = getPlayerItems.data.results.items;

 if(acceptedInventoryInstanceIds.length != playerItems.length){

   throw "your error here - acceptedInventoryInstanceIds not owned";
```

```
 }

 // we should also check if the item defId is the same as requested by the initial trade
offer //

 const requestedDefIds = new Set(tradeLog.requestedCatalogItemIds.map(item =>
item.defId));

 const matchedItems = playerItems.filter(item => requestedDefIds.has(item.defId));

 if(matchedItems.length != tradeLog.requestedCatalogItemIds.length){

   throw "your error here - acceptedInventoryInstanceIds does not match requested
item(s)";

 }



 // [5] - Deliver items to player //

 acceptedInventoryInstanceIds.forEach(itemId => {

   let giveItemResp = userItemsProxy.giveUserItemTo(tradeLog.initiatedBy, itemId, 1, 1,
true);

   if (giveItemResp.status != 200) {

     // your error here //

     // throw or return error;

   }

 });

 tradeLog.offeredInventoryInstanceIds.forEach(itemDef => {

   let giveItemResp = userItemsProxy.giveUserItemTo(playerId, itemDef.itemId, 1, 1,
true);

   if (giveItemResp.status != 200) {

     // your error here //

     // throw or return error;

   }

 });

  // [6] - Update the trade log //

 tradeLog.status = "ACCEPTED";

 tradeLog.acceptedBy = playerId;

 const updateLogResp = customEntityProxy.updateEntity(entityType, entityId,
entity.version, tradeLog, null, logTTL);

 if (updateLogResp.status != 200) {
   // your error here //

   // throw or return error;

 }

  // [7] - Send message to players //

 const sendMessResp = messagingProxy.sendMessage([ playerId, tradeLog.initiatedBy ],
tradeLog);

 if (sendMessResp.status != 200) {

   // your error here //

   // throw or return error;

 }


 response.tradeLog = tradeLog;

 return response;

}


main();
```

We can test this using one of the trade log IDs created earlier. Ensure you have a valid item ID for the player accepting the trade and that both the current player and the player who initiated the offer are using items configured as tradable in the item catalog menu.


API Explorer

Summary

Profile ID: dd633ee1-970e-4329-bad7-2d4e87a7cc31
Session ID: 2h34s39q1l44scvuhaq45rhrkp
API Domain: api.braincloudservers.com
Bypass App Enabled Check: ✓

API

Service: Script
Operation: acceptTrade
Description: Accepts the trade on behalf of the player

Parameters

```
{
  "tradeId": "93fd4671-74ae-4209-bd30-5ae93f13d57f",
  "acceptedInventoryInstanceIds": [
      "97e19283-9ba3-4de9-9792-5d37002d7cdd"
      ]
}
```

Logs

```
          "createdAt": 1751023084042,
          "updatedAt": 1751541451849,
          "version": 4,
          "stackable": false,
          "consumable": false,
          "uses": null,
          "coolDownSecs": 0,
          "recoverySecs": 0,
          "activatable": false,
          "statusName": null,
          "activeSecs": null,
          "maxActiveSecs": null,
          "statusBehaviour": null,
          "tradable": true,
          "blockchain": false
        }
      ],
      "offeredInventoryInstanceIds": [
        {
          "itemId": "d6d4b856-f39a-457d-898a-b75465bf7f8d",
          "defId": "foldablePet",
          "quantity": 1,
          "usesLeft": null,
          "coolDownStart": -1,
          "recoveryStart": -1,
          "itemData": null,
          "giftedTo": null,
          "giftedFrom": null,
          "createdAt": 1751458786154,
          "updatedAt": 1751458786154,
          "version": 1,
          "maxUses": null,
          "coolDownUntil": -1,
          "recoveryUntil": -1,
          "itemDef": {}
        }
      ],
      "allowedPlayerIds": [
        "dd633ee1-970e-4329-bad7-2d4e87a7cc31"
      ],
      "status": "ACCEPTED",
      "initiatedBy": "e65c6696-9155-433f-9806-dab57c1b36d0",
      "acceptedBy": "dd633ee1-970e-4329-bad7-2d4e87a7cc31"
    }
  },
  "success": true,
  "reasonCode": null
 },
 "status": 200
}
```

If the offer passes all the checks in the code you will see the updated trade log in the response now has the status updated and the acceptedBy field has been updated to the current player ID.

Inventory API Comparison

For quick reference, below is a table outlining all the basic PlayFab item and inventory management APIs and their corresponding requests on PlayFab.

PlayFab	brainCloud
GrantItemsToUser	AwardUserItem or PurchaseUserItem (for consuming currency)
GetCharacterInventory	GetUserItemsPage GetUserItem
RevokeInventoryItem	DropUserItem

ConsumeItem	UseUserItem or DropUserItem
OpenTrade	Custom Cloud-Code Scripts or GiveItemToUser
GetTradeStatus	Custom Cloud-Code Scripts & ReadEntity
GetPlayerTrades	
CancelTrade	Custom Cloud-Code Scripts & DeleteEntity
AcceptTrade	Custom Cloud-Code Scripts or GiveUserItem ReceiveUserItemFrom SellItemToUser