brainCloud

Upgrading to brainCloud from PlayFab

Title and User Generated Content

Table of Contents

Introduction

PlayFab allows developers to configure global content as JSON or key/value pairs, accessible by clients or internal services. These are typically used to augment or customize PlayFab features, such as reward definitions, stat multipliers, server messages, achievements, and quest definitions.

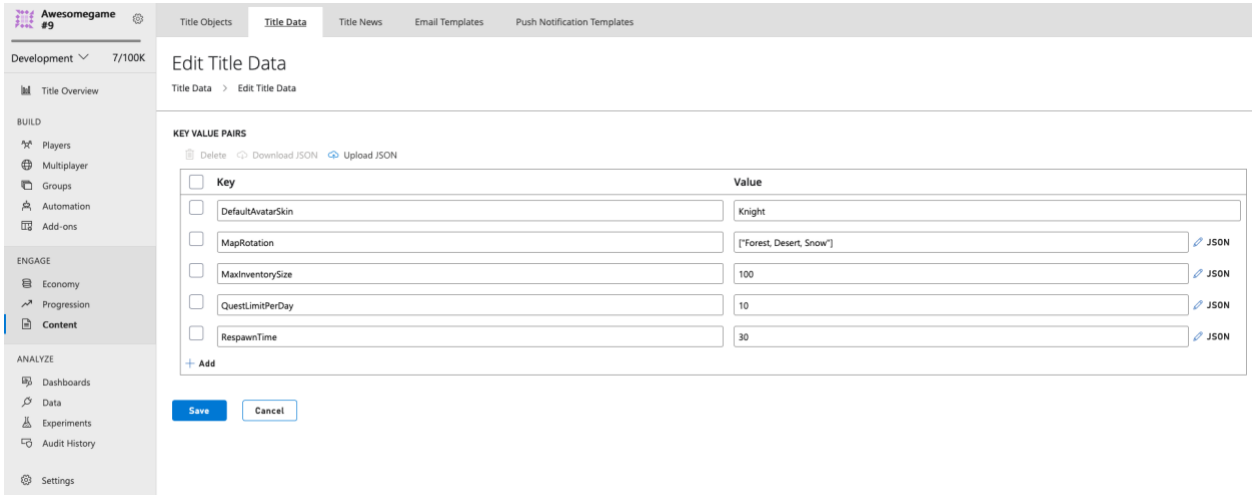
brainCloud offers similar features but expands on static global data by supporting more complex data structures and queries, enabling advanced functionality and custom features to meet your game's needs.

In this guide, we'll examine how global data is managed in brainCloud and how it can be migrated to suit your game's requirements.

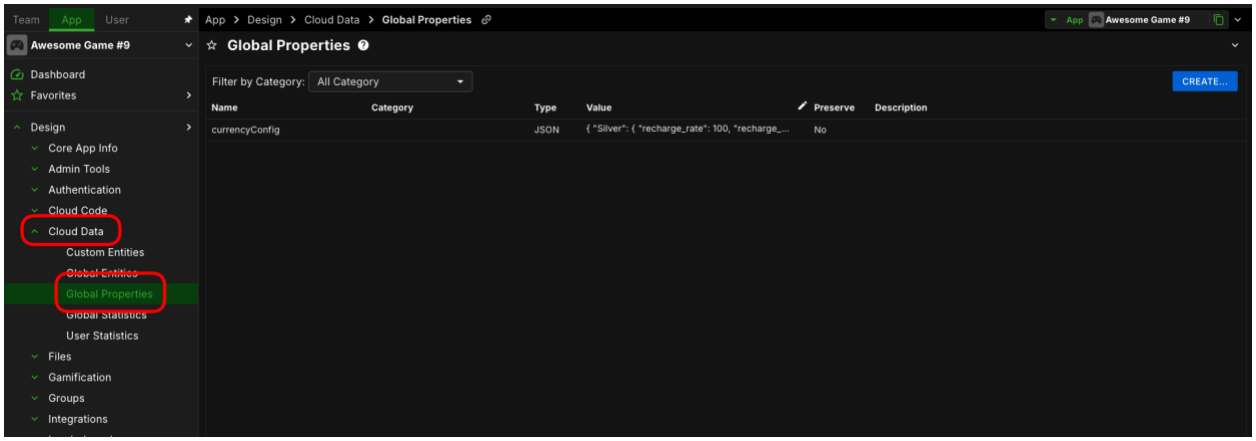
> **Note**
>
> Throughout this guide, we refer to brainCloud's "Global" meaning data available to all players without restriction, comparable to PlayFab's use of the word "Title."

Title Data

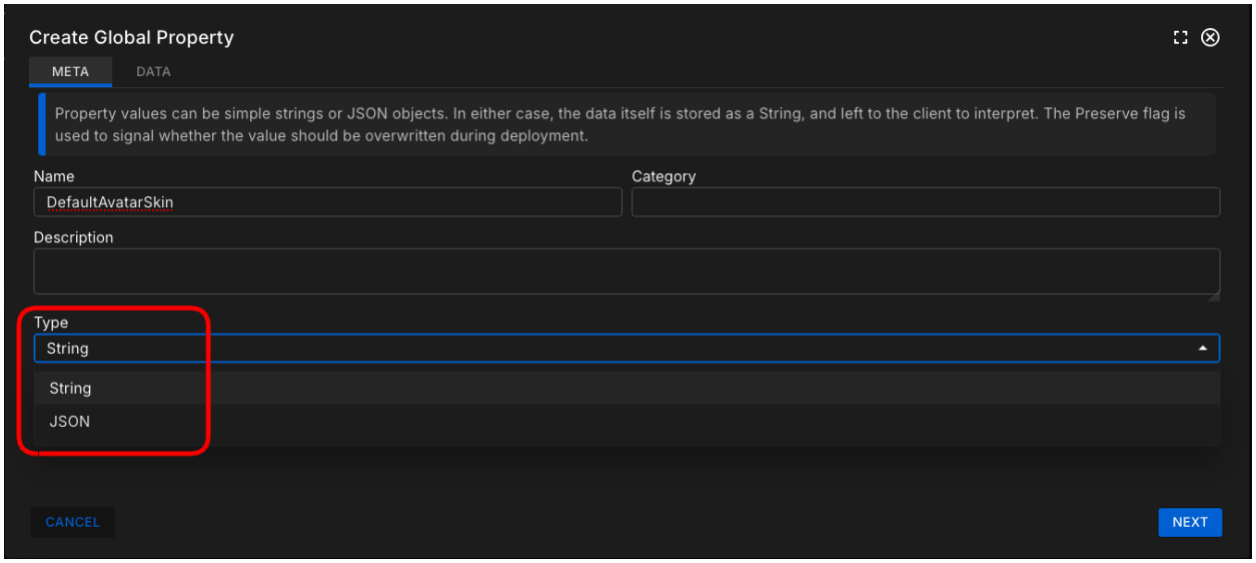
In PlayFab, Title Data enables players to access static key/value pairs that represent game configuration data.



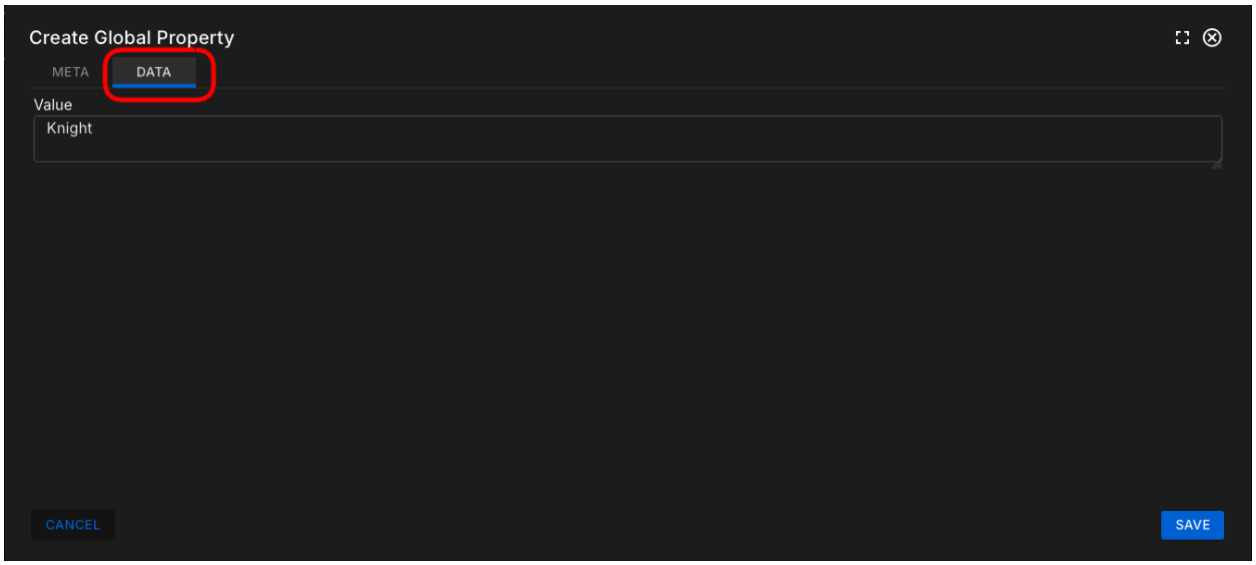
brainCloud offers a similar feature called Global Properties, found in the **Design → Cloud Data** menu.



Clicking the **Create** button displays options for creating Global Properties. For example, you can choose a string or JSON type, similar to the options available in PlayFab.



In the **Data** tab, you can enter the value for your Global Property. As in PlayFab, these properties can use either JSON or string values.



Reading Global Properties

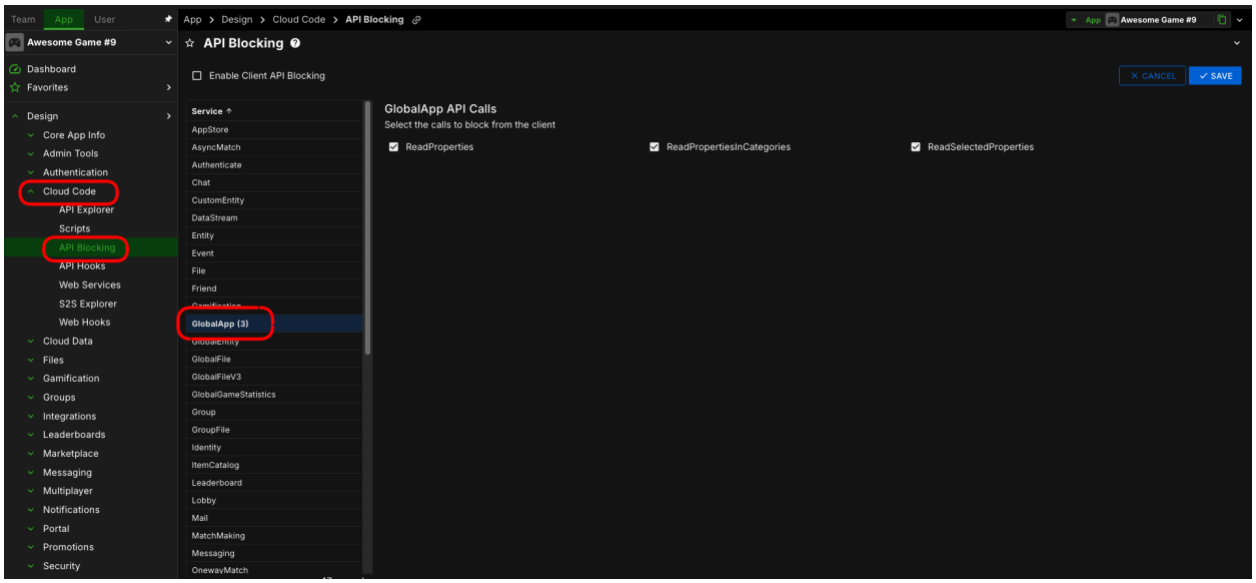
Accessing Global Properties in brainCloud is similar to retrieving Title Data in PlayFab. The ReadProperties API returns a list of all configured properties, allowing you to parse the desired property by key, as you would with the PlayFab SDK. Additionally, two other useful APIs are available: ReadSelectedProperties, which retrieves only specified properties, and ReadPropertiesInCategories, which lets you group properties by category and fetch all properties within a particular category.

Title Objects

Title Objects in PlayFab are similar to Title Data but are typically used for more complex JSON structures. They are accessible only via the server API or cloud-code. While brainCloud doesn't have a direct equivalent, Global Properties can serve as a replacement. However, since Global Properties are client-accessible, you'll need to prevent client access if your game requires server-only data. There are two ways to restrict access to these objects:

Block Global Properties API

You can block client access to any standard brainCloud API by navigating to **Cloud-Code → API Blocking** and selecting the needed service group "Global App." Blocking these requests prevents clients from accessing Global Properties while still allowing access from cloud-code.

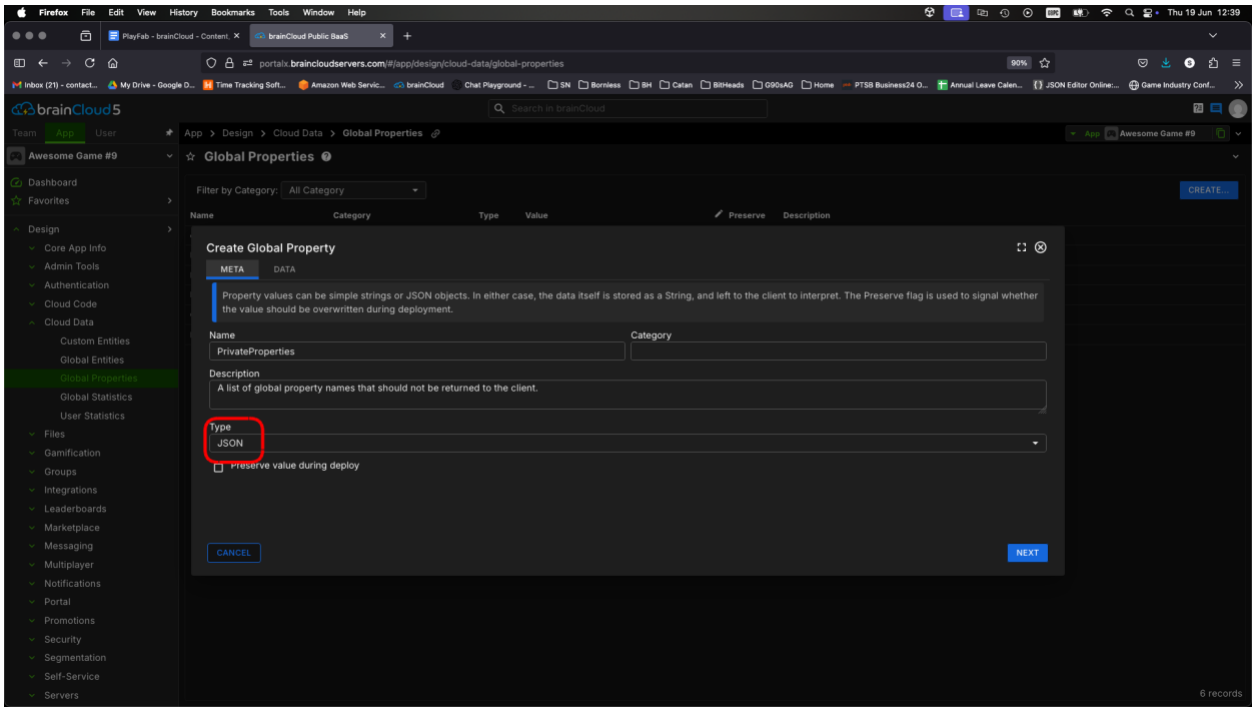


One problem with this solution is that you might be using Global Properties as a solution for migrating your Title Data, in which case you still need access to Global Properties from your client. If this is the case, let's take a look at another solution.

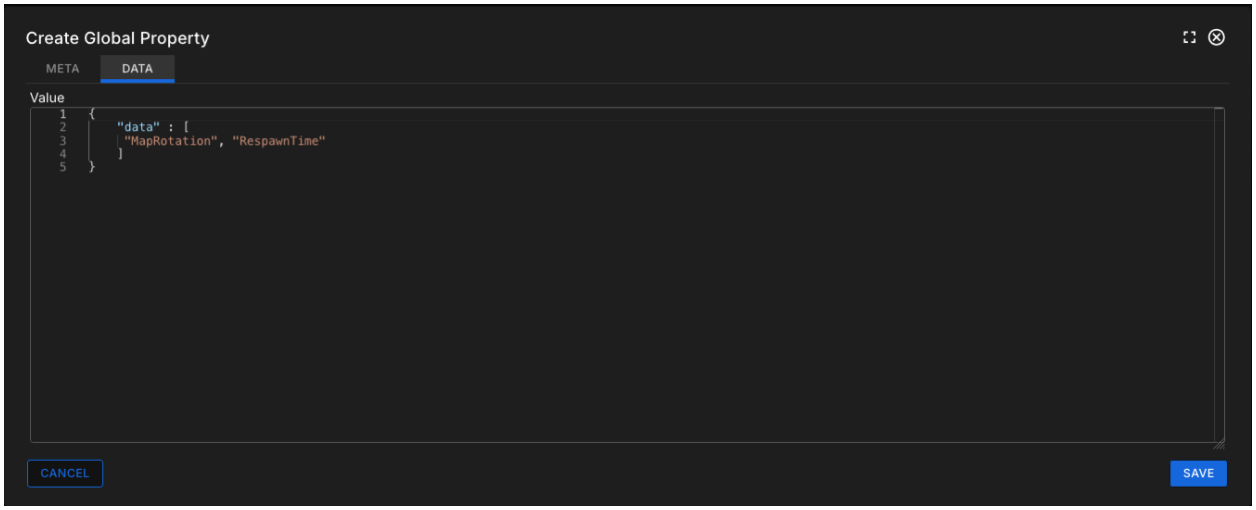
Filter Private Global Properties In Response

This approach is more complex but allows you to use Global Properties for both private and public data.

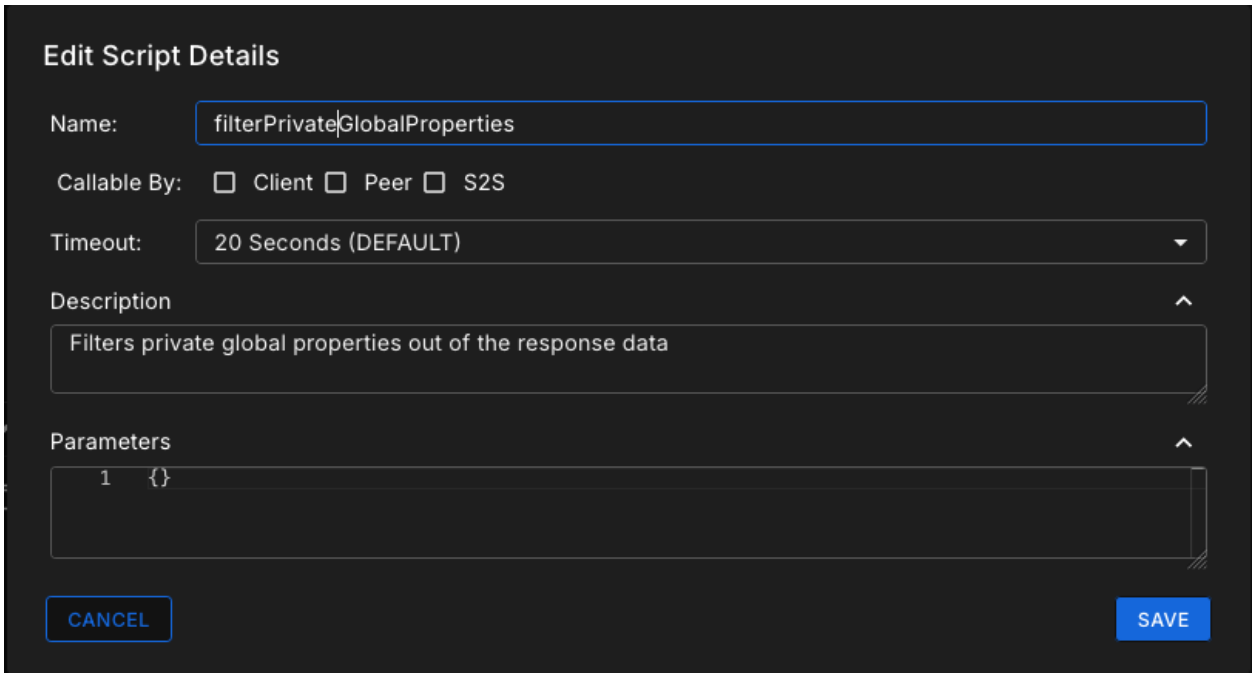
For this we will need to use custom cloud-code and API Hooks. Check out our Cloud Scripts guide for more information. We want to maintain a list of private Global Property names, and we can also use Global Properties for this. Create a new Global Property and set it as a JSON type.



For that data, add an array with the names of the properties you want to keep private.



Once saved, create a new cloud-code script called **filterPrivateGlobalProperties** and ensure this script isn't callable from the client.

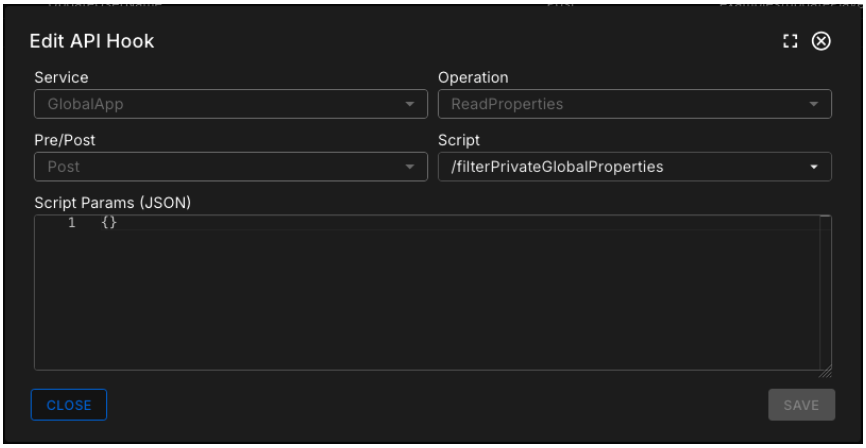


This script only requires a few lines of code. It will first parse the “PrivateProperties” value from the response, then iterate through that array to remove any listed properties. Finally, it will remove itself from the response and return the filtered array.

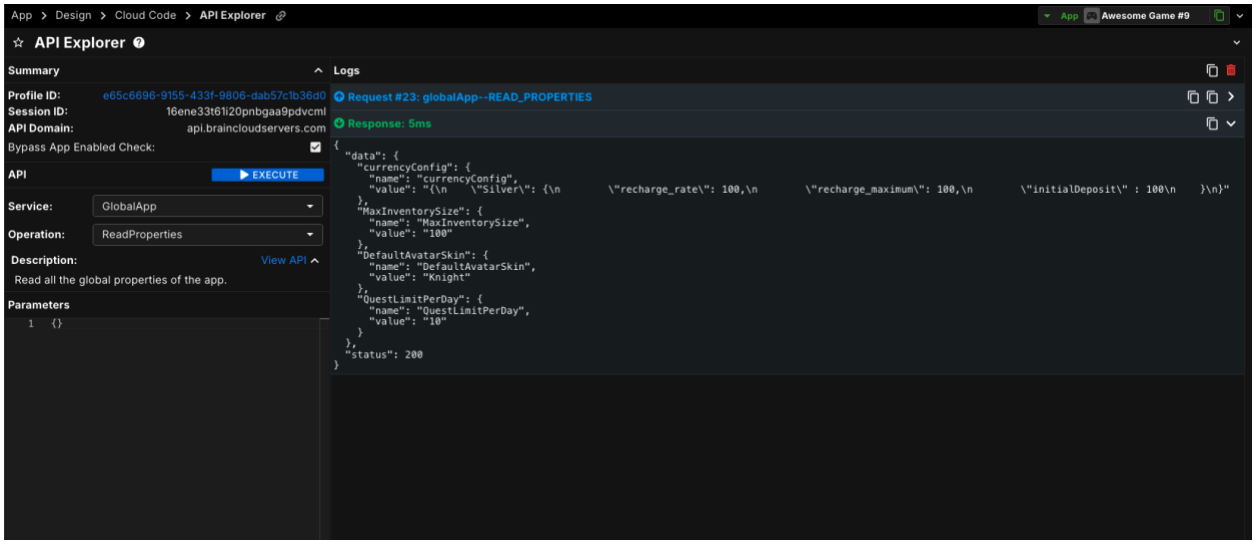
```
"use strict";


function main() {


 let response = { status: 200, data: data.message };


 // [1] - Grab the arry of private properties //
 const privatePropArr = JSON.parse(data.message.PrivateProperties.value).data;
 privatePropArr.forEach(function (propName, index) {
   // [2] - Remove this property from the response //
   delete response.data[propName];
 });
 // [3] - Remove the private property list from the response //
 delete response.data.PrivateProperties;
```

```
 return response;

}



main();
```

The last step is to add this script as a post-hook to the **ReadProperties** request.



You can test this using the API Explorer (see our Cloud Scripts guide for instructions). Our example includes some pre-configured Global Properties that will be filtered out. If your script works, you should see that the “PrivateProperties” list is absent from the response.



Note

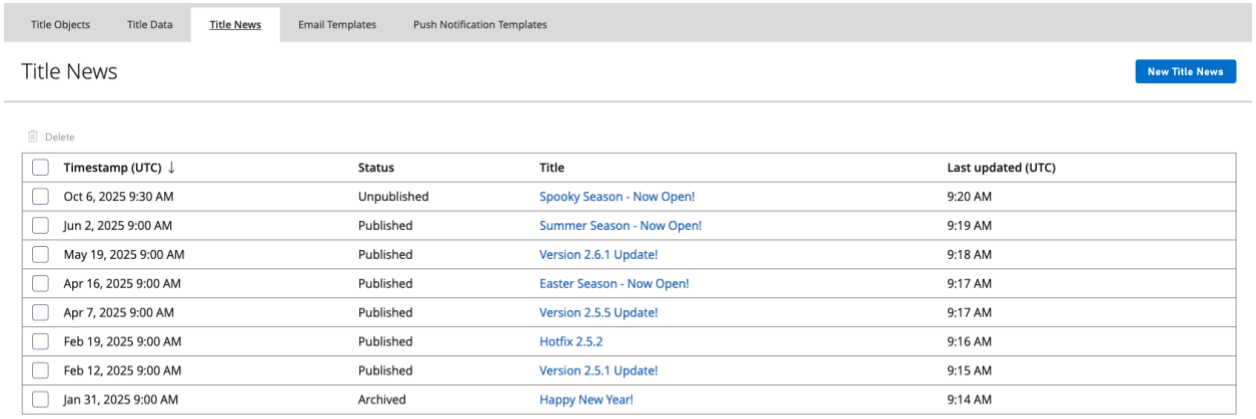
There are three client APIs for reading properties: **ReadProperties**, **ReadSelectedProperties**, and **ReadPropertiesInCategories**. You may need to adapt this example and apply the filter to each API to ensure your properties remain secure.

Title Internal Data

Title Internal Data is another type of global data stored on PlayFab. While similar to Title Data in structure, it is accessible only from cloud-code. If you need to migrate Title Internal Data, option 2 from the example above (filtering Global Properties) is suitable for keeping your data hidden from clients while still allowing access through Global Property APIs via cloud-code.

Title News

In this guide, we've covered how to migrate static "Title" data and objects from PlayFab to brainCloud. However, brainCloud includes other types of global data that may be more flexible depending on your needs. Let's look at how one of these features could serve as a migration path for Title News in your game.

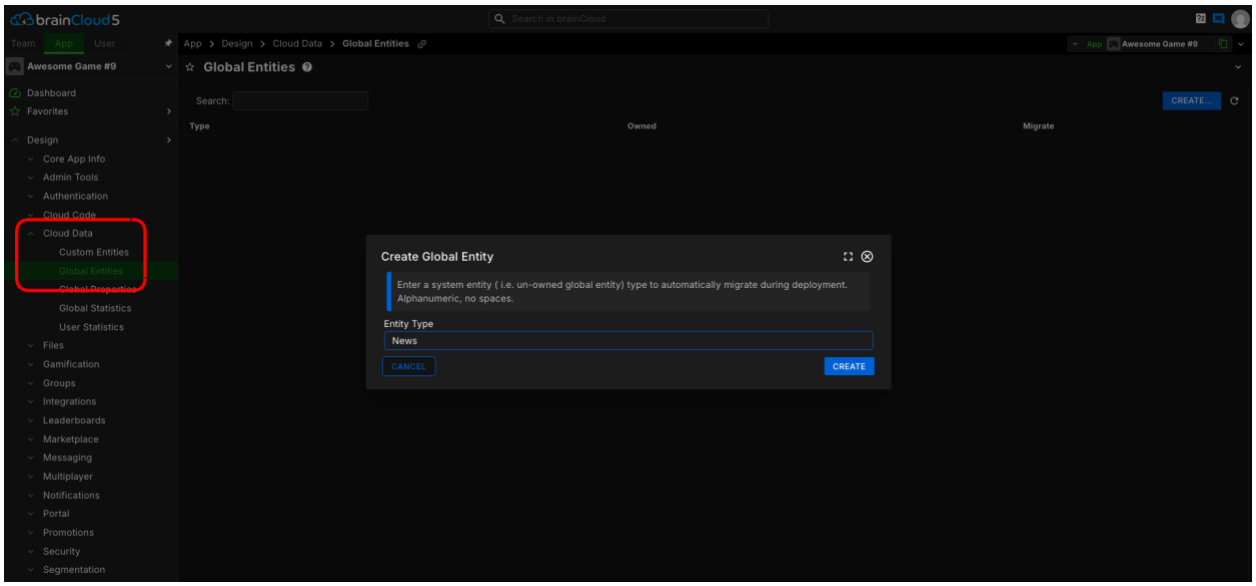


Timestamp (UTC) ↓	Status	Title	Last updated (UTC)
Oct 6, 2025 9:30 AM	Unpublished	Spooky Season - Now Open!	9:20 AM
Jun 2, 2025 9:00 AM	Published	Summer Season - Now Open!	9:19 AM
May 19, 2025 9:00 AM	Published	Version 2.6.1 Update!	9:18 AM
Apr 16, 2025 9:00 AM	Published	Easter Season - Now Open!	9:17 AM
Apr 7, 2025 9:00 AM	Published	Version 2.5.5 Update!	9:17 AM
Feb 19, 2025 9:00 AM	Published	Hotfix 2.5.2	9:16 AM
Feb 12, 2025 9:00 AM	Published	Version 2.5.1 Update!	9:15 AM
Jan 31, 2025 9:00 AM	Archived	Happy New Year!	9:14 AM

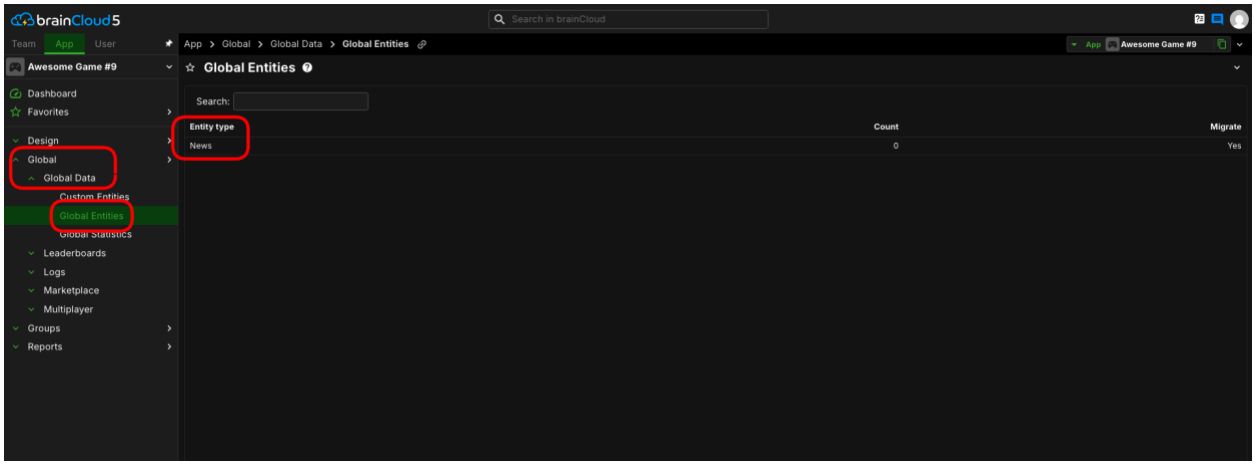
Global Entities

Global Entities allow developers to configure globally available JSON documents stored in a MongoDB database. This lets you create a list of documents within a collection (for example, replicating PlayFab's Title News collection) and provides the ability to query, list, and sort these documents as needed.

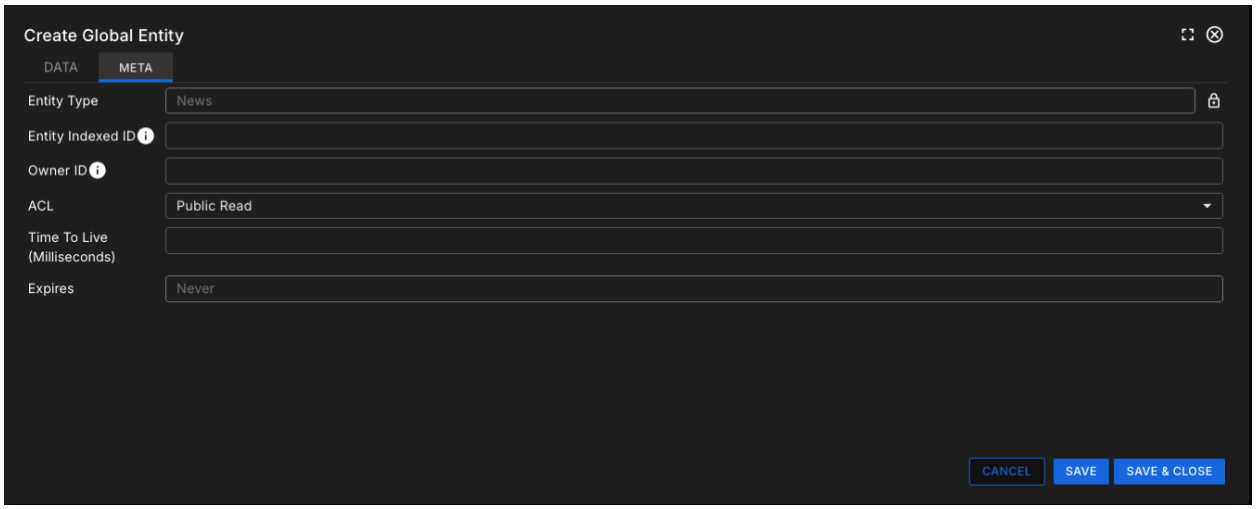
To set up a new Global Entities collection, go to the **Cloud Data → Global Entities** menu and create a collection called "News."



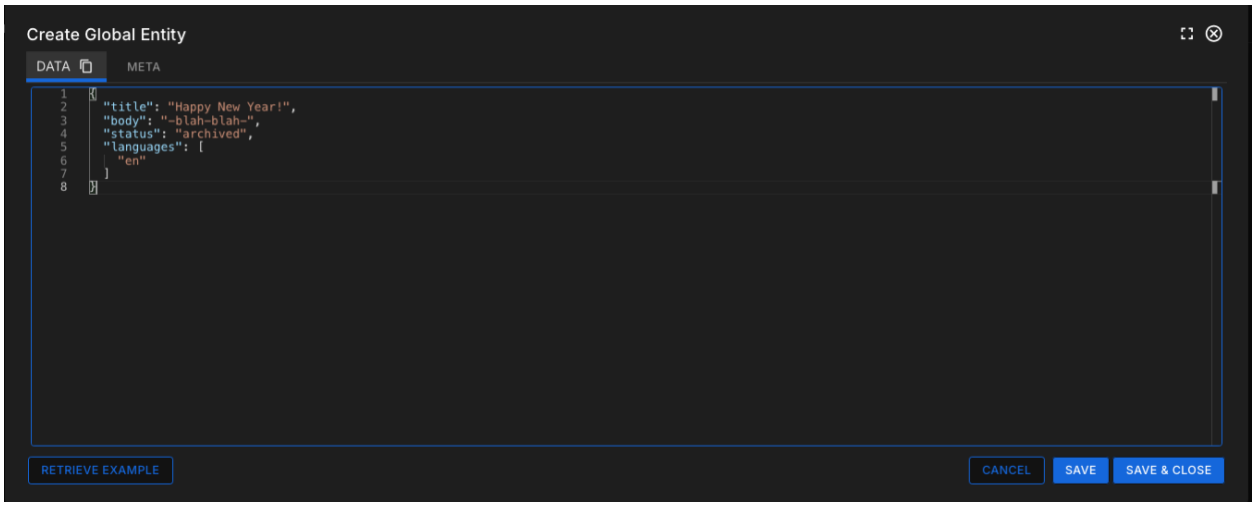
Next, populate the collection by going to **Global → Global Data → Global Entities** and clicking on the "News" collection.



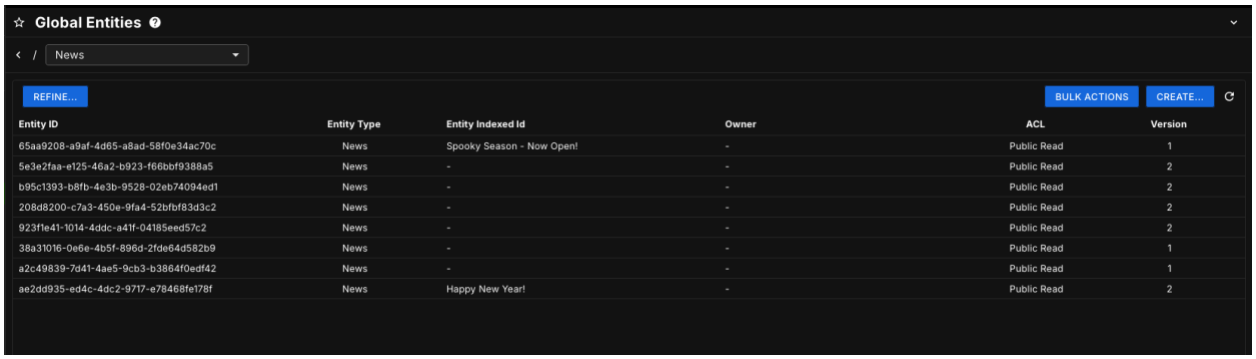
From this view, you can browse and query all the documents in your collection. Since there are no documents yet, let's add some. Click the **Create** button in the top right corner to add a new document.



You'll notice a few parameters you can set, such as TTL, owner ID, and index ID. For this example, we don't need to change any of these. News should be publicly readable and doesn't belong to a player. Indexing new entries isn't necessary, as we usually retrieve lists of posts rather than a specific one. Click the **Data** tab to add your first news post example.



If you save this example now, you'll see it appear in your collection list.



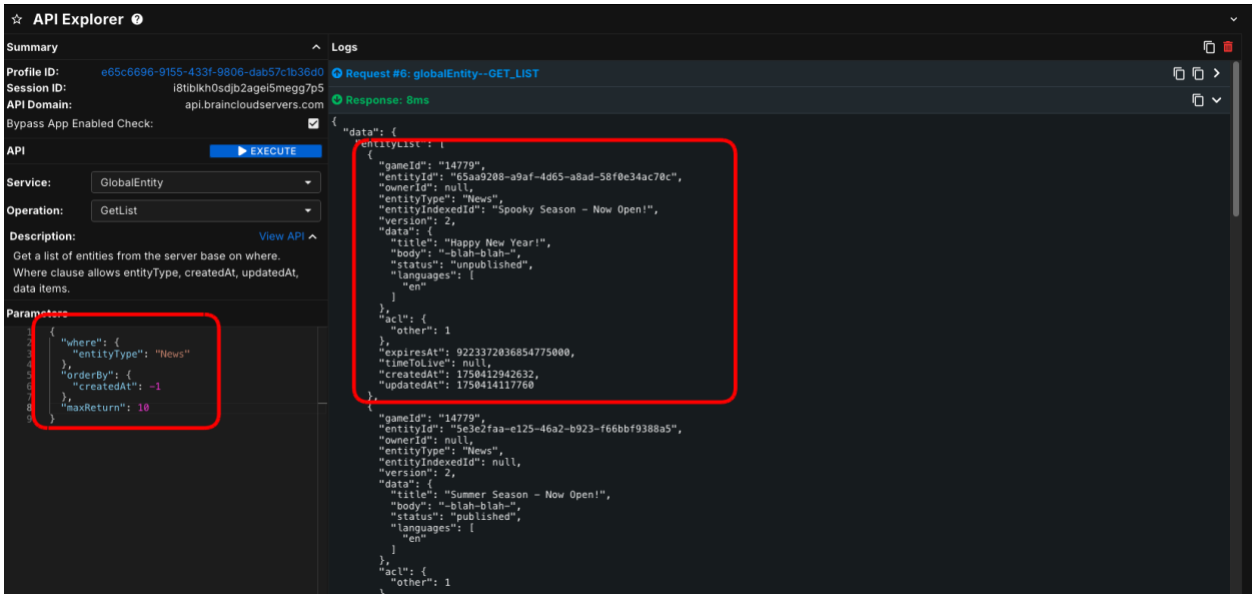
Note that we included two examples using the news post title as the index ID to make searching and updating records easier, but this isn't required. As long as the collection doesn't grow very large, this approach won't cause performance issues.

Now, let's look at how to filter this list by querying from the client. You can test this in the API Explorer using the Global Entity service with the GetList operation.

Three useful queries for news are: getting the latest news, retrieving all published posts (excluding unpublished or archived ones), and filtering news by language.

Get Latest News

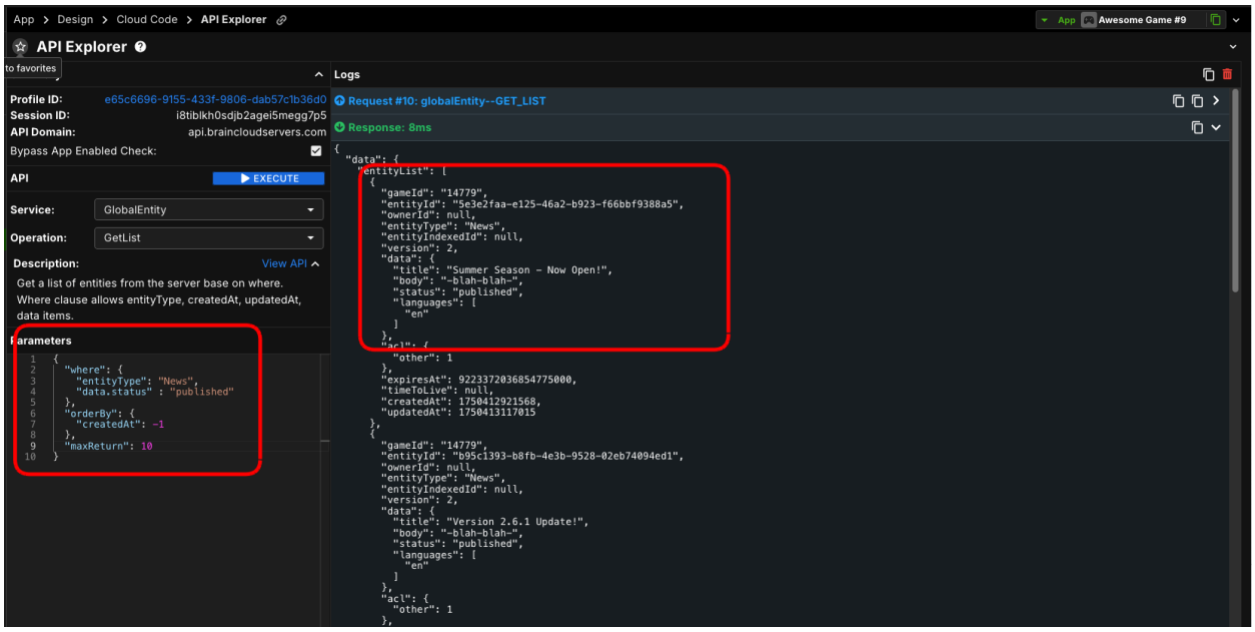
In this example, we simply order by the `createdAt` field to retrieve the latest posts. However, you'll notice this also includes unpublished posts, which isn't ideal. In the next example, we'll filter those out.



Also note the **maxReturn** parameter. You can use this to limit how many posts are returned. For example, to retrieve only the latest news post, set **maxReturn** to 1.

Get All Published News

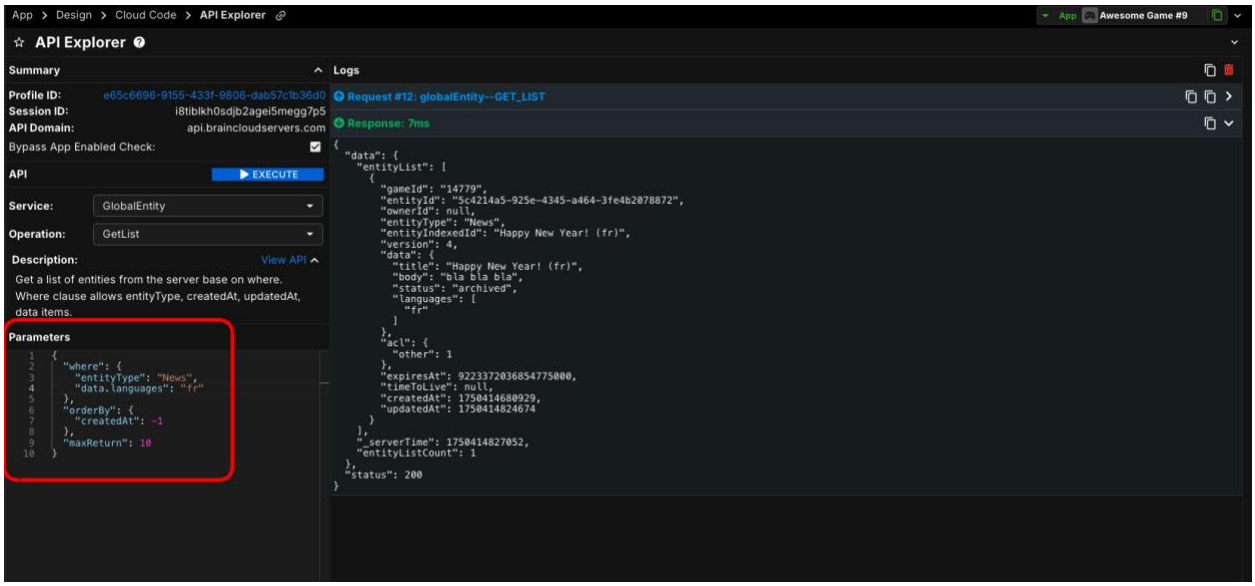
For this, you'll need to reference the "data" object of the document, where the JSON data is stored.



Here, we're selecting only documents where "data.status" is set to "published." As a result, the first and last posts in the collection are absent from the response, since their statuses were set to "unpublished" and "archived," respectively.

Filter By Languages

This is a slightly more complex query. For the example, I created another document that includes localization.



Hopefully, these examples illustrate how Global Entities can be used not only to replicate Title News during migration, but also to create more complex features. This can speed up migration or help you deliver new features to your players after the move.

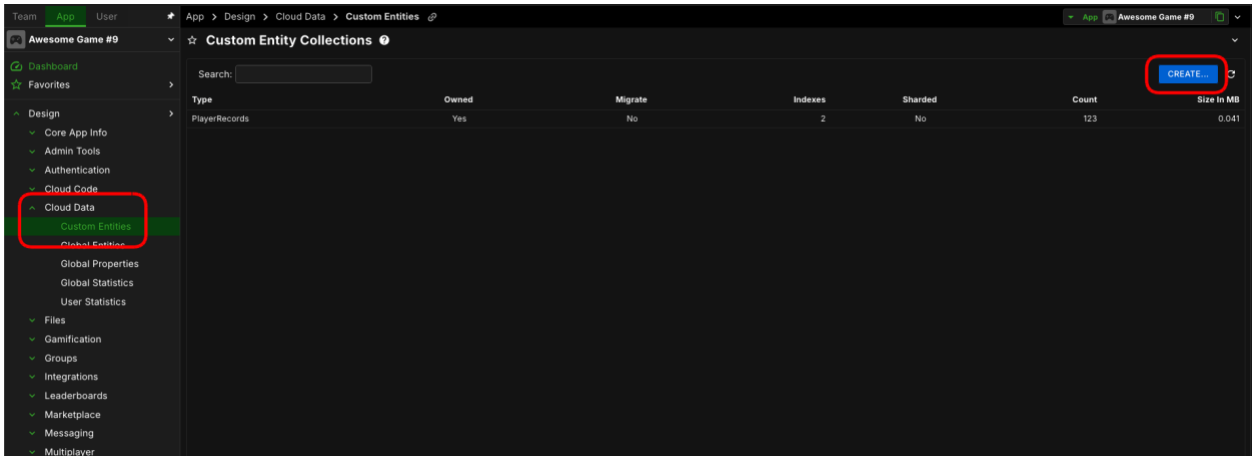
User Generated Content

PlayFab's User Generated Content (UGC) feature enables players to create custom items using the Catalog v2 API. UGC allows files to be uploaded and attached to these items, and lets other players search for, report, or delete items. This makes UGC a flexible solution for various custom content needs. Let's look at how these abilities can be reproduced in brainCloud. In this example, we'll explore how users can upload custom levels to brainCloud, allow other players to search for them using different parameters, download them, and report levels to admins.

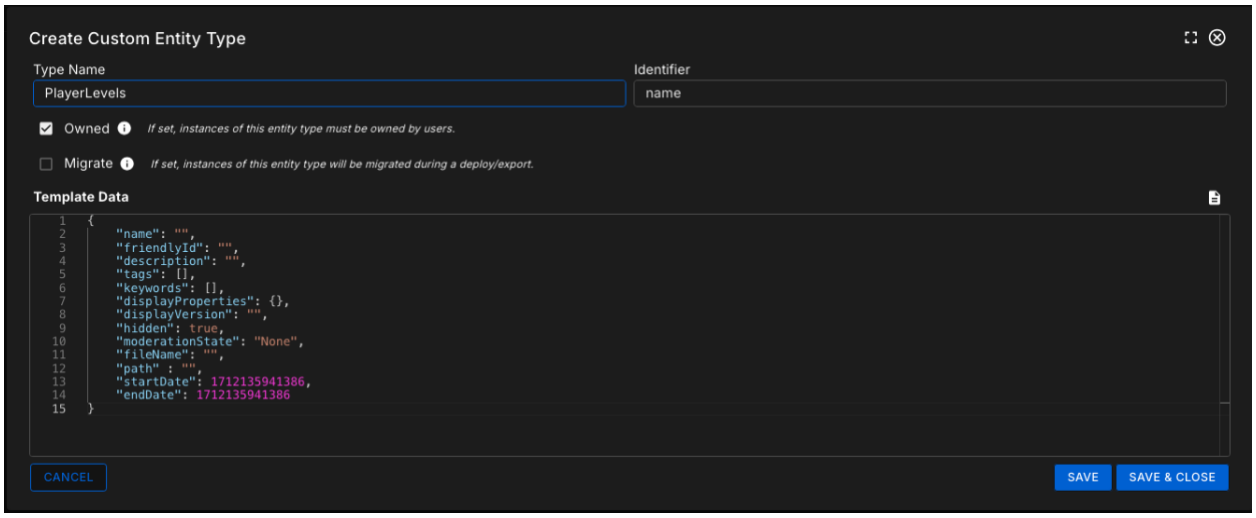
Custom Entities

To achieve this we will be using Custom Entities. Custom Entities are similar to the Global Entities used previously for Title News. They allow you to create custom MongoDB collections, store documents, and run complex queries on that data. Custom Entities are highly flexible and can be used to create unique game features or extend existing brainCloud functionality based on your needs.

For this example, we'll create a new collection called "PlayerLevels." Go to **Cloud Data** → **Custom Entities** and click the **Create** button in the top left of the dashboard.



This collection will use the level name as the identifier, ensuring each level name is unique. To match PlayFab Catalog v2 items, we'll add some of those parameters to the template for this collection.



Submit A Custom Level

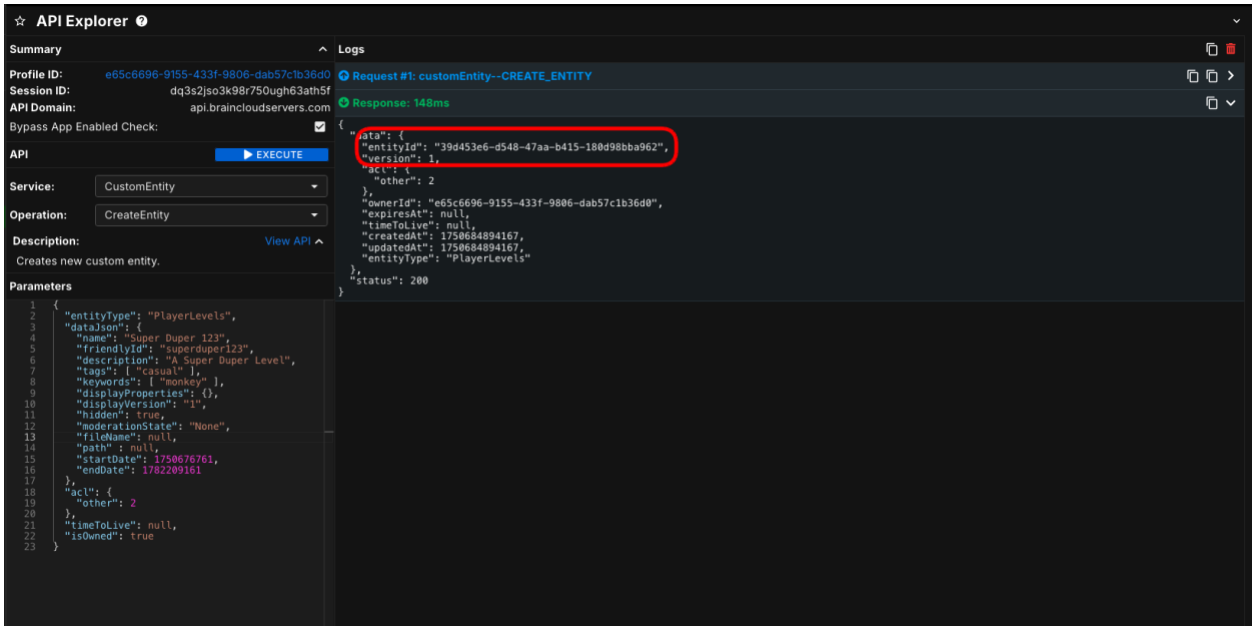
Next, we want players to be able to submit custom levels to this collection. The CreateEntity request, called from the client, handles this. However, let's look at how to test and debug this process from the portal using the API Explorer.

Submit the following JSON payload using the CustomEntity service and **CreateEntity** operation.

```
{
 "entityType": "PlayerLevels",
 "dataJson": {
   "name": "Super Duper 123",
   "friendlyId": "superduper123",
   "description": "A Super Duper Level",
   "tags": [ "casual" ],
   "keywords": [ "monkey" ],
   "displayProperties": {},
   "displayVersion": "1",
   "hidden": true,
   "moderationState": "None",
```

```
    "fileName": null,
    "path" : null,
    "startDate": 1750676761,
    "endDate": 1782209161
  },
  "acl": {
    "other": 2
  },
  "timeToLive": null,
  "isOwned": true
}
```

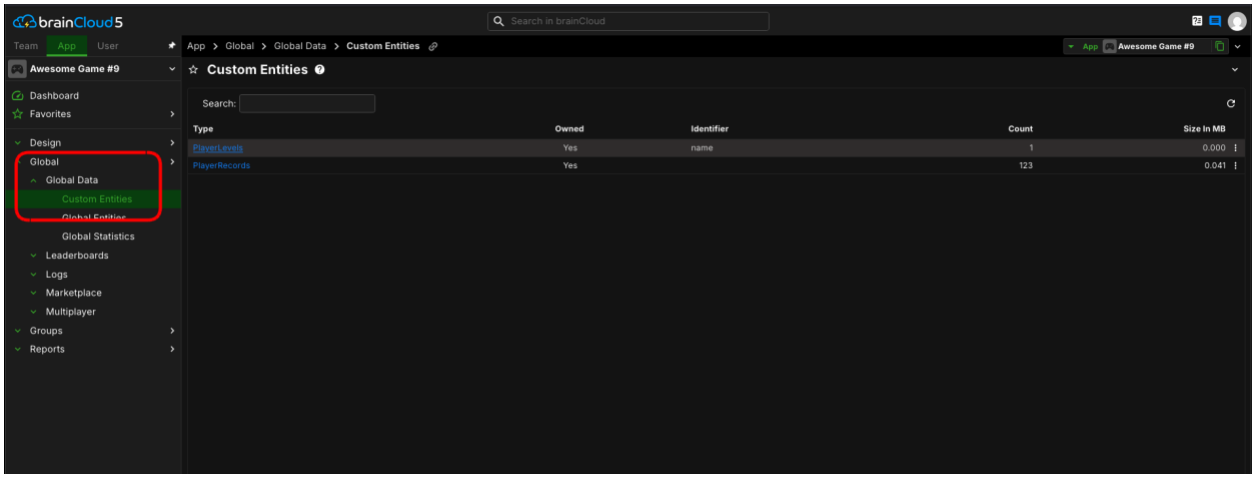

The key values here are **name**, used as the document's identifier, and the access-control-list (ACL), which determines whether other players can read or update the document (in this example, other users need update access for moderation). The **isOwned** field is also set to 'true'.



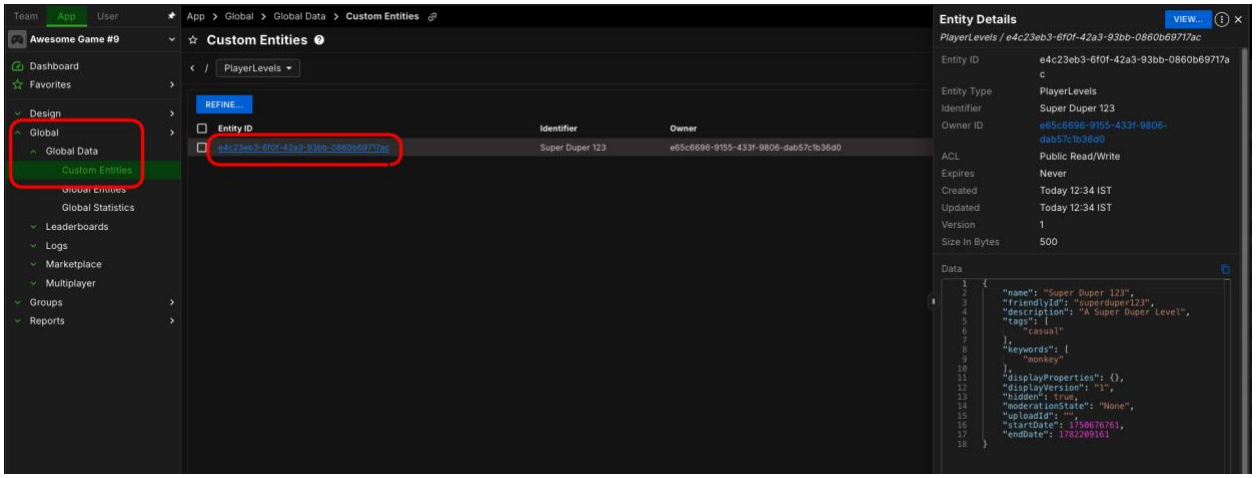
In the response, you'll see the entityId of this document. Make sure to note it for future requests.

Viewing Collection Documents

You can view collections in the portal by going to **Global → Global Data → Custom Entities** and click on the **PlayerLevels** collection.



There should be only one document in this collection. You can inspect it by clicking the **Entity ID** highlighted in blue. A panel on the right side of the dashboard will display all information about the document.



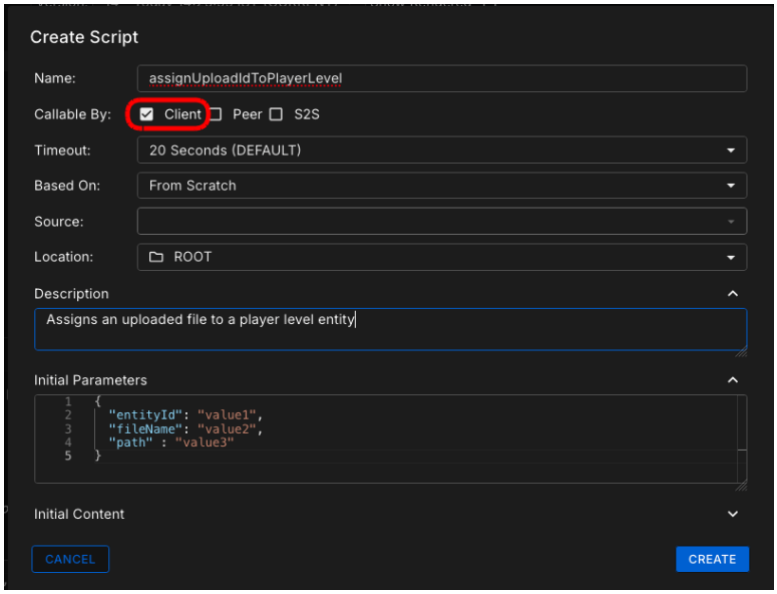
Uploading Level Binaries

The next step is to upload the level data for this custom entity. Like PlayFab, you need a separate request to upload the binary content and then link it to the custom entity.

Use the UploadFile request to upload your level binary. This action can only be done from the client, so it can't be tested through the portal. Once uploaded, you'll receive the **fileName** and

path in the response. We will assign the **uploadId** to the custom entity document using custom cloud-code and API Hooks. (Check out our guide Cloud Scripts for more details.)

Create a new cloud-code script called **assignUploadIdToPlayerLevel**. Make sure this script is callable from the client.

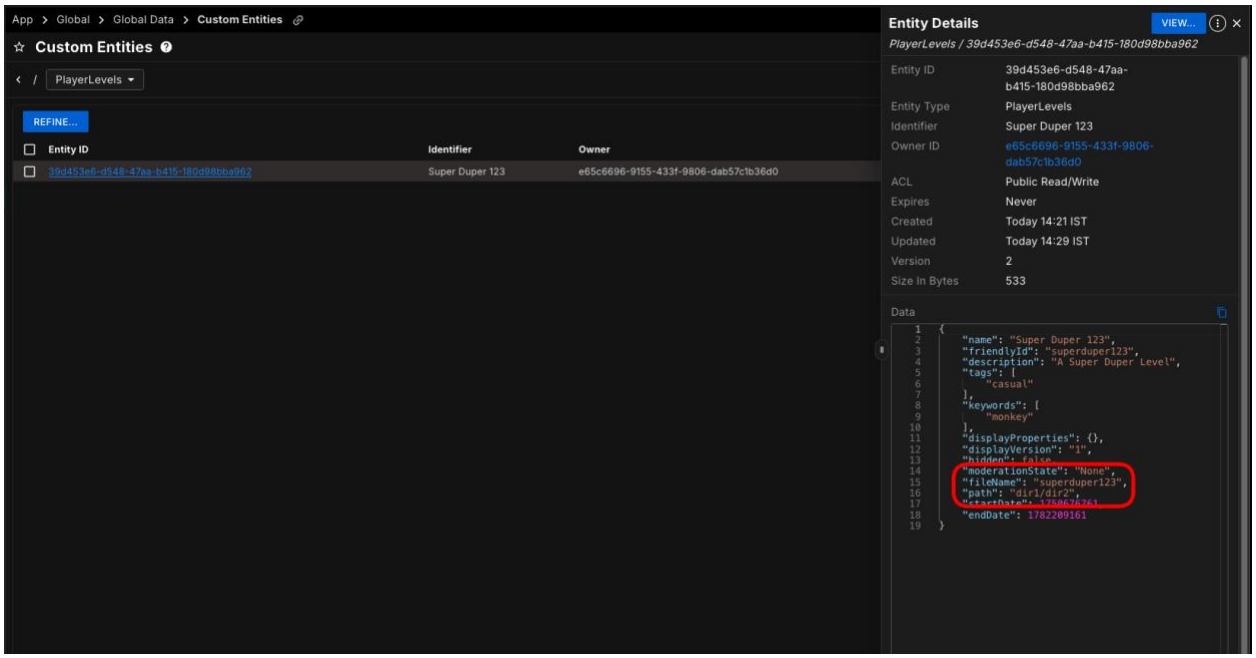


We'll provide a quick example for this script. You may need further checks or validation for your own game.

```
"use strict";


function main() {

 var response = {};


 // [1] - Load the entity //

 const entityType = "PlayerLevels";

 const entityId = data.entityId;

 const customEntityProxy = bridge.getCustomEntityServiceProxy();

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

 if (getEntityResult.status == 200) {


   // [2] Validation - return an error if the uploadId doesnt exist //
```

```
   //                - return error if this player is not the owner //

   // -> your code here



   // [3] - Update the entity document //

   const currentVersion = getEntityResult.data.version;

   const fieldsJson = {

     "path" : data.path,

     "fileName" : data.fileName,

     "hidden": false               // << we automatically unhid the level when it has a
file attached

   }

   const updateEntityResult = customEntityProxy.updateEntityFields(entityType, entityId,
currentVersion, fieldsJson);

   if (updateEntityResult.status != 200) {

     // return errors //

   }



 } else {

   // return errors - if the entity doesnt exist, return an error //

 }



 return response;

}



main();
```

Using the path and fileName from your uploaded file, along with the entityId, you can test this in the API Explorer. To confirm the update, check the fields in the collection viewer.



Similar code to what was shown in this example could be used to let players moderate content. Deleting content can be handled with the DeleteEntity API, provided you first validate that only the owner is able to delete their own content.

Searching Documents

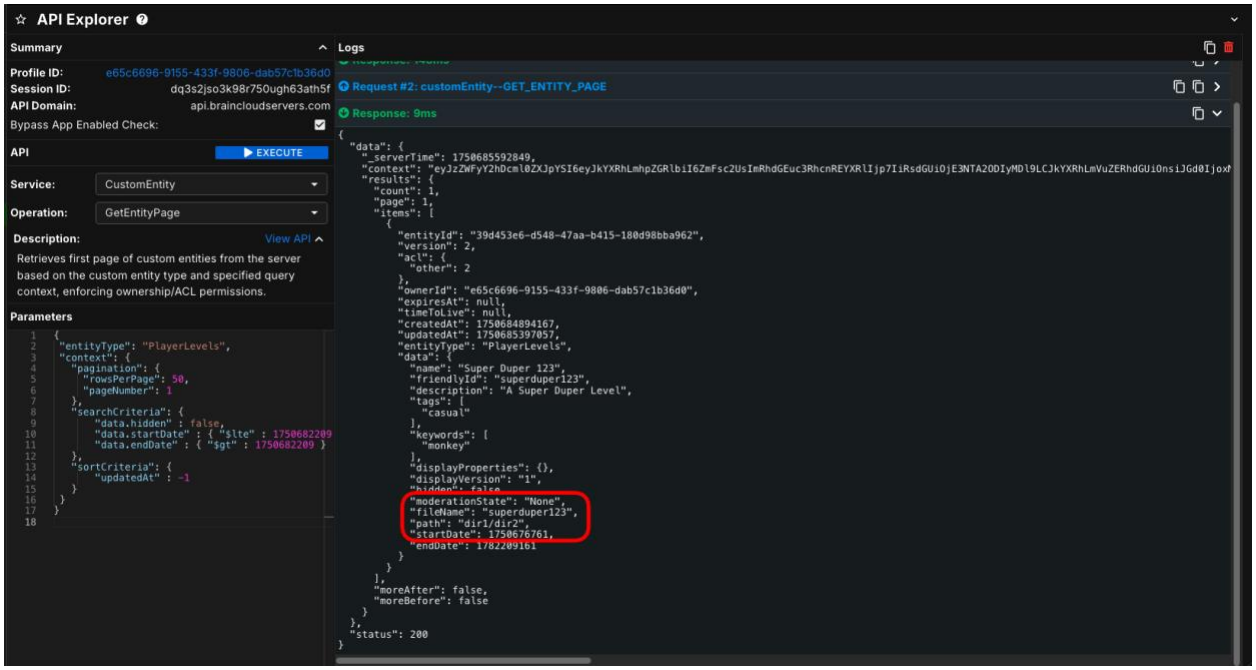
Next, we'll look at how to search and filter documents using the GetEntityPage request, which works similarly to the **GetList** request from the Global Entities example.

To start, we want to retrieve the latest levels that aren't hidden and fall within the start and end dates. You can test these queries in the API Explorer.

```
{
 "entityType": "PlayerLevels",
 "context": {
   "pagination": {
     "rowsPerPage": 50,
     "pageNumber": 1
   },
   "searchCriteria": {
       "data.hidden" : false,
       "data.startDate" : { "$lte" : 1750682209 },
```

```
        "data.endDate" : { "$gt" : 1750682209 }

    },

    "sortCriteria": {

        "updatedAt" : -1

    }

  }

}
```

The key part of this query is the search criteria, which filter by the hidden parameter and the start and end date. The sort uses the updatedAt field, with "-1" indicating newest first.



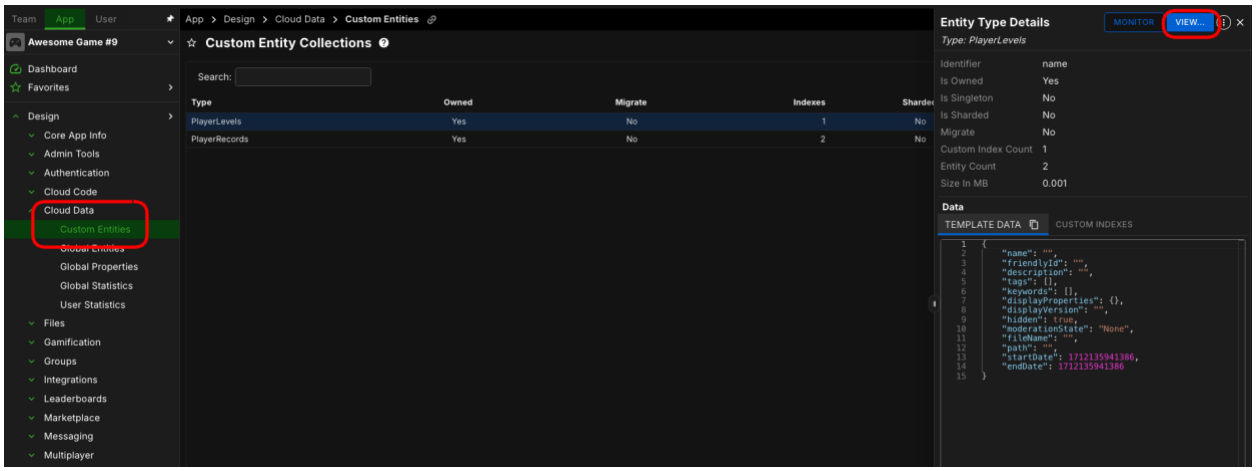
Take note of the fileName and path parameters returned from this request, as you'll use them later to download the file.

Sorting & Searching By Name

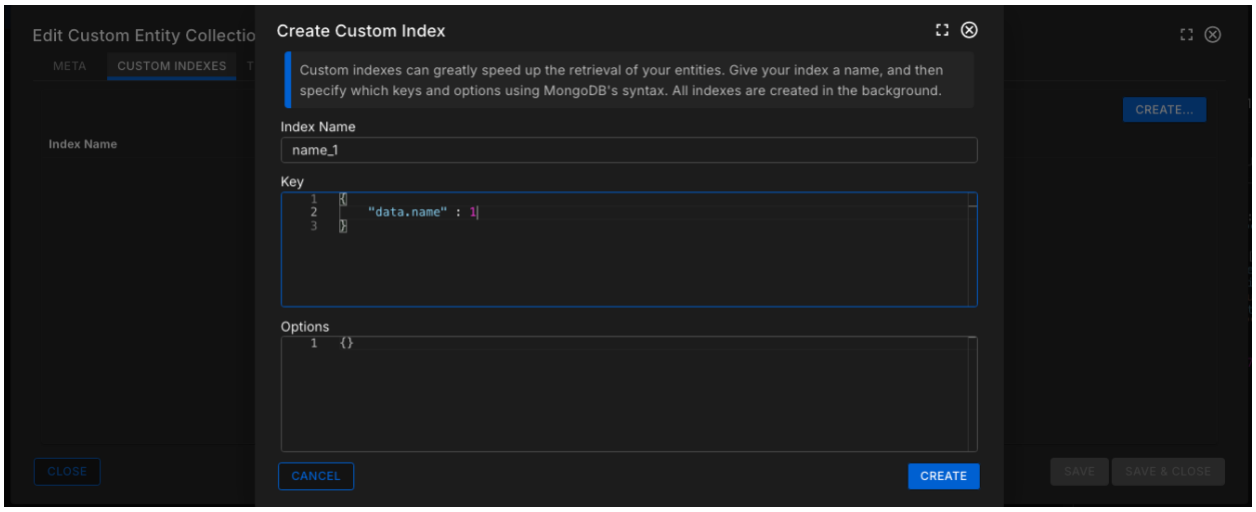
You can sort alphabetically by using the name field as the sort criteria. However, it's recommended to add a custom index before performing more complex queries or sorts on custom entities.

Custom Indexes

For this example, we'll create a custom index on **data.name** to improve sorting and querying efficiency. To do this, go back to your collection configuration, click on the collection, and then click the **View** button.

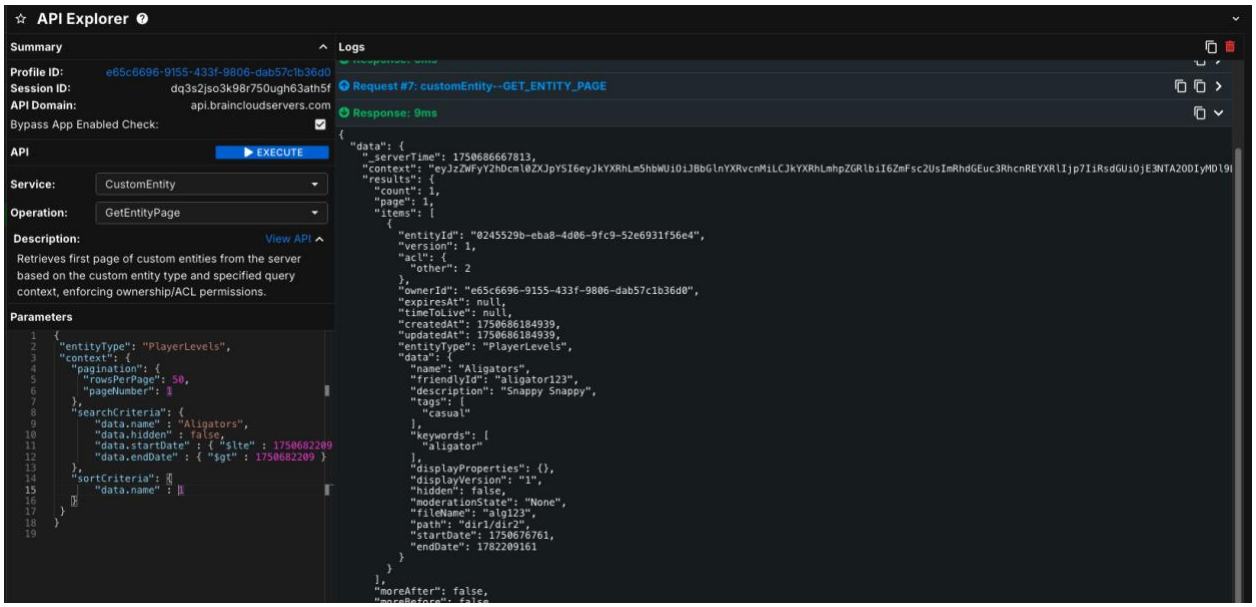


Next, click on the **Custom Indexes** tab and then the **Create** button. Name the index "name_1" and add the key as shown in the image below.

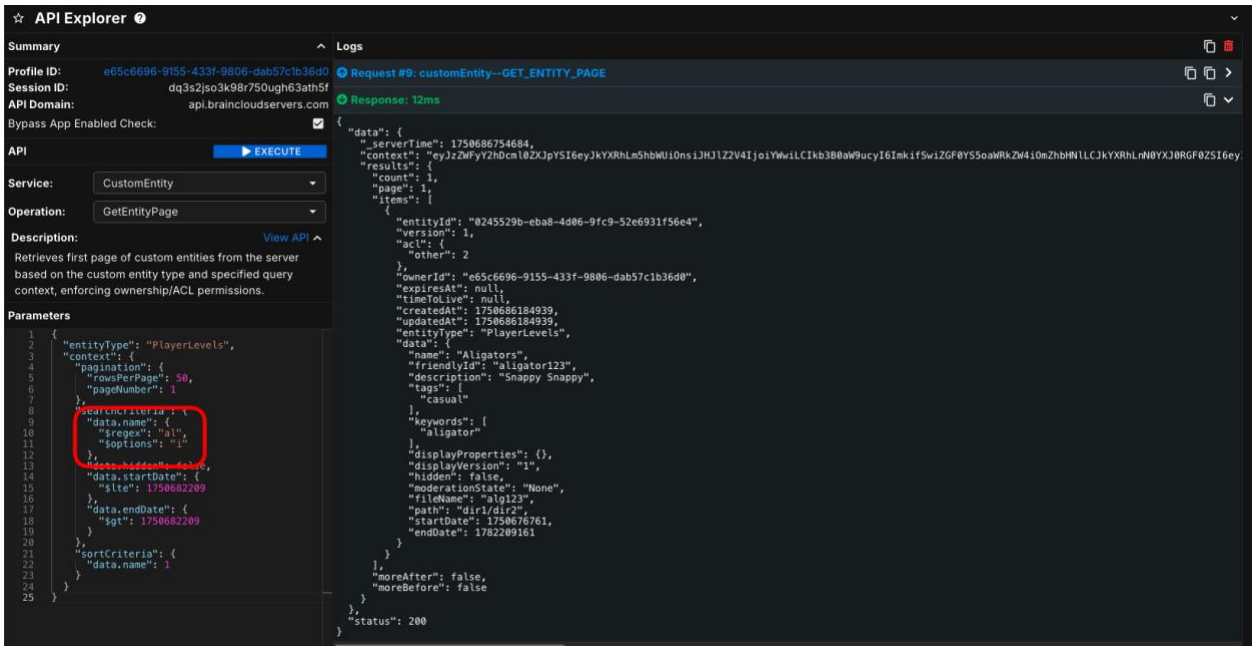


Now that the index has been added, you can use the GetEntityPage request to query it. You can switch between alphabetical (1) and reverse alphabetical (-1) order.

Another common approach is to search by name or partial name. This is easily done by adding a line to the search criteria for the **data.name** parameter.



Or, you can use a regex query for a partial search.



Downloading Files

Because the GetEntityPage response contains the path and fileName you can use the GetCDNUrl request to get a short-lived URL (lasting 1 hour) where you can download the file.

User Files

While the v1 API of PlayFab only allowed for storing player data as key/value pairs—similar to the title data (see migration guide here <link>), v2 introduced functionality for uploading and downloading binary files.

brainCloud also provides a CDN feature which includes both Global Files and User Files.

Generally, brainCloud's Files system works like PlayFab but you do not need to initiate the upload with a separate request. You can upload files directly using a single UploadFile call. There are then options for cancelling, and checking upload progress based on the **uploadId** returned from this request.

As with PlayFab, you can view user files in the brainCloud User dashboard by navigating to **Data → User Files**.

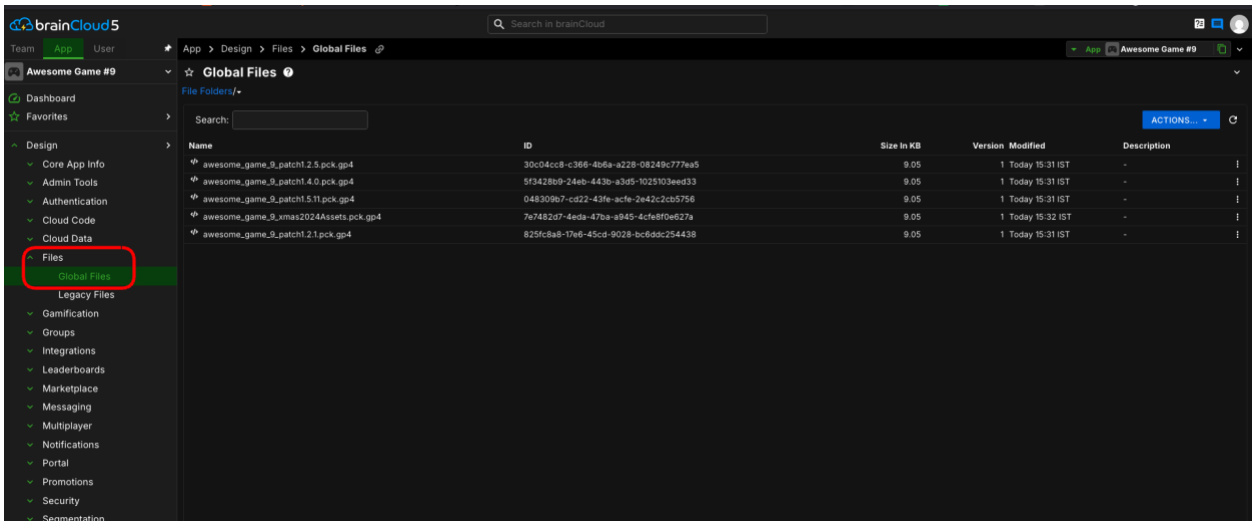
You can list or download user files by using ListUserFiles to get the file name and path, and then GetCDNUrl to generate a download link.

It's important to note that brainCloud User Files can be set to shareable (public) or non-shareable (private). This feature allows file sharing between users without additional features or custom code.

Global Files

Global Files are typically used for content accessible to all users, such as asset updates or DLC. PlayFab doesn't have a built-in feature for uploading files directly to the portal, though it can be done with a workaround using User Generated Content and a hard-coded entityId.

With brainCloud, this isn't necessary. Developers can upload binary content through the portal, making all files accessible via the Global Files API. You'll find Global Files in the **Files → Global Files** menu.



Before finishing this section, it's important to note that brainCloud offers more generous feature limits for its File/CDN feature. The table below compares these restrictions.

	PlayFab v1	PlayFab v2	brainCloud
Binary Files	✗	✓	✓
File Size	10Kb	10Mb *(Up to 100Mb depending on pricing plan)*	20Gb

Limit	300 *(per player)*	10 *(depending on pricing plan)*	none
Folder Support	✗	✗	✓

If your game requires migrating CDN and files, or you use an external CDN, you may be able to leverage brainCloud's Files feature. Consider contacting brainCloud support for advice on migration and how ETL can be performed.