How to use

Simple Account with Database Asset

New Line

Last update: 04/04/2020 (version 1.0.1)

Stay up to date, click here

Introduction

Attention : This asset use an external service, you need to have a PlayFab SDK in your project, otherwise, this asset doesn't work.

This PlayFab SDK is normally preinstalled in the asset, however, if you encounter errors, you can install it here.

If you already have this sdk, we invite you to remove it from your project to integrate ours, and thus avoid duplicates or possible errors.

Get Started

Step 1 : Here is the first thing to do when you use this asset. You must imperatively have the playfab SDK in your project. This is a service that will be used to store all player data.

Normally this one is attached in the asset, but if you encounter errors delete it and download it here.

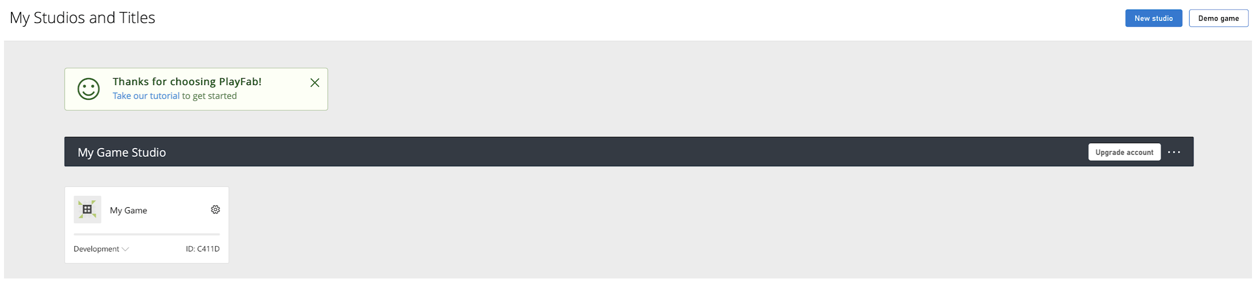
PlayFab Website   PlayFabSDK

Step 2 : Once you have the sdk, you must create a PlayFab account. Go here.

For instructions on how to use this service, refer to page 13 of this manual.

Configurations

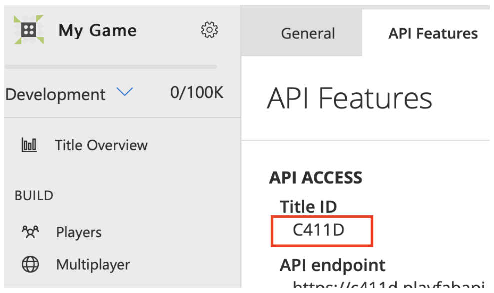
Step 1 : Congratulations, you are now connected to the Play Fab services, all you have to do is create your application, to do so, simply follow the steps indicated. You should have this :



Step 2 : Now click on your game to access this information, then follow this video.

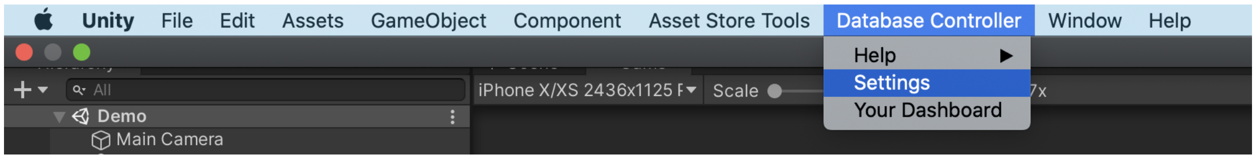
This step is very important, it will allow users to store their data. WARNING by checking this box, your game can be submitted to cheat! If you want to switch to a secure transfer, this <image xlink:href="https://lh3.googleusercontent.com/Ej7KhaQ5gfaHCoXSAxii6Xw6Kvznf29mqHpGs...AVrzQ7OfsnWqiddPCYAAhDv-WVJKv7hkqgQwXSu6O5Svv4Rokb8Wd8obn6lFVSZRl2HnpOiNJg" width="100%" height="100%" preserveAspectRatio="none" transform="scale(0.0005)" transform-origin="0 0">option is available in the Complete Account with Database Asset, which also contains a virtual currency system, friends list, leaderboards, and more....

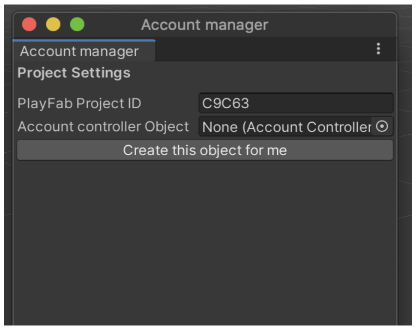
Step 3 : Now, all you have to do is retrieve your game ID, in the settings, from where you were :



Copy your Title ID to your clipboard and proceed to the next step.

Step 4 : And so, now you can go back to Unity to set up your project. In the action bar, go to Database Controller > Settings




Account manager

Project Settings

PlayFab Project ID: C9C63

Account controller Object: None (Account Controller)

This window should then open.

To understand the next steps, you can go to the demo scene included in the asset.

Step 5 : Go to the scene you will use for your connection system. *ATTENTION only one scene can be used per project to avoid any problems.*

Then, in the settings window that was opened earlier, click the "Create this object for me" button. After which, you can enter your application ID *(step 3)* in the Project ID selection.

If you are developing a mobile game, and would like to use the anonymous connection option, please switch your project to iOS or Android if you haven't already done so. *This option is only available for mobile.*

Now all you have to do is fill in all the necessary objects in the management window and press "Apply". A confirmation message should appear in the Unity console.

Congratulations, you have successfully configured your project. Now you can build your level in order to implement the login system for the user. You can help yourself from the demo scene to better understand.

Implementations (for Unity C#)

Note : During this part, you will be able to see some implementations that you may not use in your project, if so, just skip the part in question.

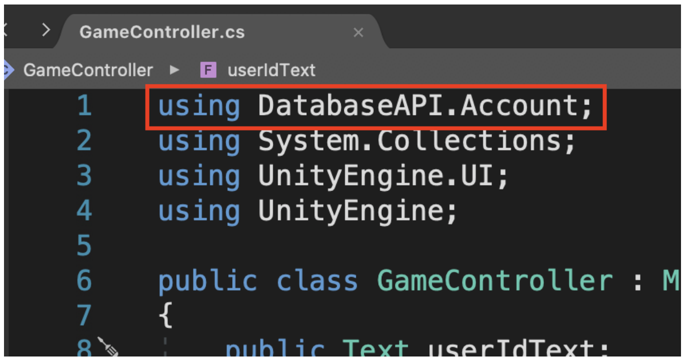
WARNING: do not use the traditional connection (part 2 of the implementation), and at the same time the anonymous connection (part 3 of the implementation)

1 - Get Started

In order to properly prepare your system, you will need only one thing, a script that will manage the elements of your scene.

If you have a problem or hesitation, do not hesitate to consult our demo script, or join us on discord.

Step 1 : Open your script, and at the top, insert the following line : 'using DatabaseAPI.Account;'



```
using DatabaseAPI.Account;
using System.Collections;
using UnityEngine.UI;
using UnityEngine;

public class GameController : M
{
    public Text userIdText;
```

(this is our demo script, you can find it in the asset folder > demo > scripts).

Now, you're able to use our asset in your script.

2 - Connection

Step 1 : Once the API is imported into your script, you can create your first calling functions.

Here are the available instructions:

1 - Get the username of the person to be registered *(Complete void bellow)*

```
public void GetUserName(string value)
{
    AccountController.controller.GET_USER_USERNAME(value);
}
```

2 - Get the email address of the user to connect/register *(Complete void bellow)*

```
public void GetUserEmail(string value)
{
    AccountController.controller.GET_USER_EMAIL(value);
}
```

3 - Get the password of the user to log in/register *(Complete void bellow)*

```
public void GetUserPassword(string value)
{
    AccountController.controller.GET_USER_PASSWORD(value);
}
```

4 - Send a connection request (line 107 in demo script)

```
AccountController.controller.LOGIN_ACTION();
```

5 - Send an account creation request (line 109 in demo script)

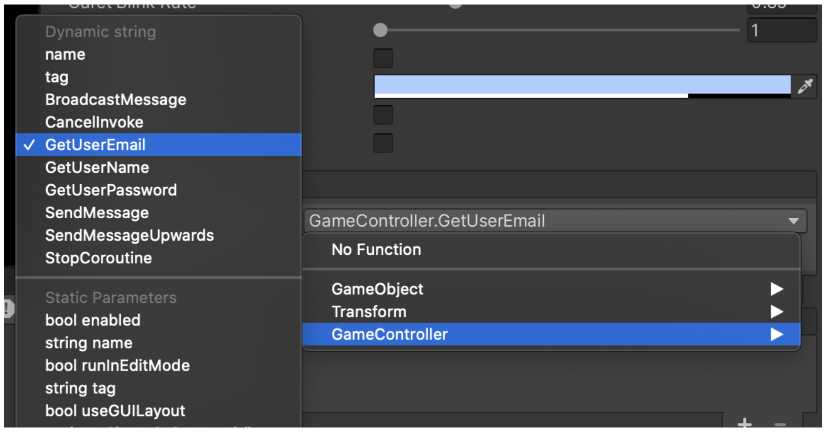
```
AccountController.controller.ON_CLIC_CREATE_ACCOUNT();
```

6 - Send a Logout Request (line 113 in demo script)

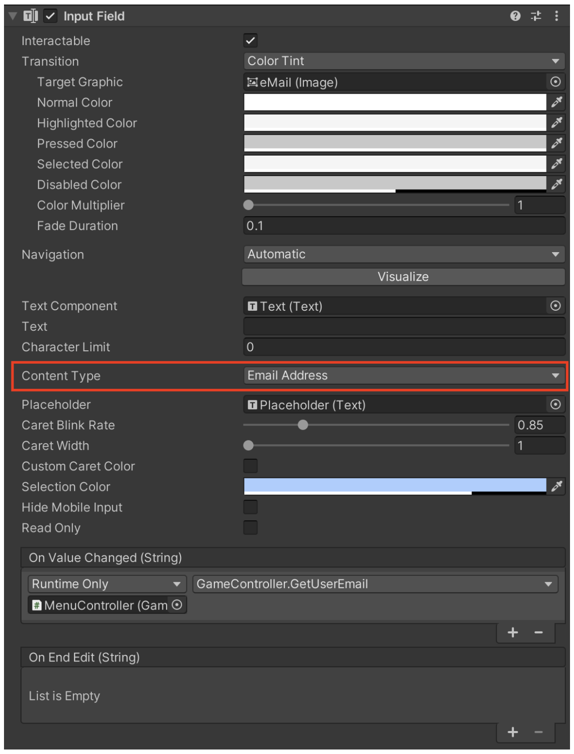
```
AccountController.controller.LOG_OUT();
```

Step 2 : All right, now, save and go back to unity to link these new functions to your buttons.

The first 3 functions listed above must be added to [InputFields](), in the “On Value Changed” section.



CAUTION choose the Dynamic string, otherwise your function will not work.


Input Field

Interactable ✓
Transition: Color Tint
Target Graphic: eMail (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Visualize
Text Component: Text (Text)
Text
Character Limit: 0
Content Type: Email Address
Placeholder: Placeholder (Text)
Caret Blink Rate: 0.85
Caret Width: 1
Custom Caret Color
Selection Color
Hide Mobile Input
Read Only

On Value Changed (String)
Runtime Only | GameController.GetUserEmail
MenuController (Gam

On End Edit (String)
List is Empty

ATTENTION (yeah, again...) be careful, it's not necessary, but you can select the true option in Content type, Email Address for email, Password for the password, and standard for username.

Now, you just have to put the login or connect function to your button in the on Click function.

3 - Mobile anonymous connection

If you are here, it’s because you are developing an app for mobile, fine.

This anonymous login allows the user to start playing without creating an account. The anonymous login still create an account to the user, which will allow him an automatic login every time the application opens. Its ID and password will be defined by its phone ID. Using the anonymous connection does not prevent linking your profile to an existing account or a new one afterwards.

Note : If you want to use this option, you don’t need a connection scene, you are strongly advised to do this section in your main menu scene.

The following steps can be found in the demo script, from line 116 to 127.

Prerequisites: in your scene, main menu for example, you will need a button to open the backup account section, and a panel that will allow the user to enter this data (email, and password as well as a username, which will only be processed if the connection does not work).

Step 1 : Write the new functions in your script

1 - This first command is simply used to open your connect recovery panel.

```
public void OpenCloseRecoverySection()
{
    AccountController.controller.OPEN_RECOVERY_PANEL();
}
```

2 - Now, this function is used to request a recovery connection, like on a login button

```
public void CreateRecoveryAccount()
{
    AccountController.controller.ON_CLIC_ADD_RECOVERY();
}
```

4 - User data

Well, you’re almost done with this asset. There’s only one thing left to do, save your player’s data.

This section is relatively simplistic, everything is already done for you.

The first thing to know, is that when connecting, all user data is stored in a local dictionary. You can view it and retrieve this data by simply typing

```
int tempValue;

AccountController.playerData.TryGetValue(NAME OF THE VALUE SAVED, out VALUE TO UPDATE)
```

You can see this template in the demo script, line 44.

How to save new data or update one:

```
AccountController.controller.SetStat(NAME OF THE VALUE TO SAVED, VALUE TO SAVE);
```

You can see this template in the demo script, line 76.

WARNING! However, you can only save integer (int) values!

Extras

In this last part of the Asset manual, you can find several small functions that may be useful to you.

1 - Knowing when you are logged into an account

You can say that this option has no interest, but on the contrary, it’s necessary! When you log in, how do you know if that connection failed, and therefore how do you make sure that you can start the next scene, such as a main menu.

To access this information, you have this boolean available:

```
AccountController.controller.connectedToAnAccount
```

If you are logged in, this boolean will return True.

If you are looking for how to integrate it, you can consult the demo script of line 34 to 41.

2 - Know when data has finished loading

Looking at these lines 34 to 41 of the demo script, you must have seen this:

```
AccountController.controller.getStatsFinished
```

This statement is also a boolean. It allows to know the state of loading of the user’s data. This boolean will equal True when all data is finished loading.

This option allows for example the use of a loading screen, to avoid that once logged in, the user sees erroneous data because not yet updated.

3 - Retrieve errors

It’s also one of the most important elements, how to warn the user of a failure.

To do this, you have access to a variable that will allow you to directly retrieve the entire error message:

```
AccountController.DebugMessage
```

this simple command allows you to retrieve the error message, then you can either process it or return it as it was to the user.

Example of use:

```
public Text errorMsg; // using UnityEngine.UI;

void Update()
{
    errorMsg.text = AccountController.DebugMessage;
}
```

4 - Get the User ID

Do you need your player’s server ID? you can retrieve it by typing

```
AccountController.userID
```

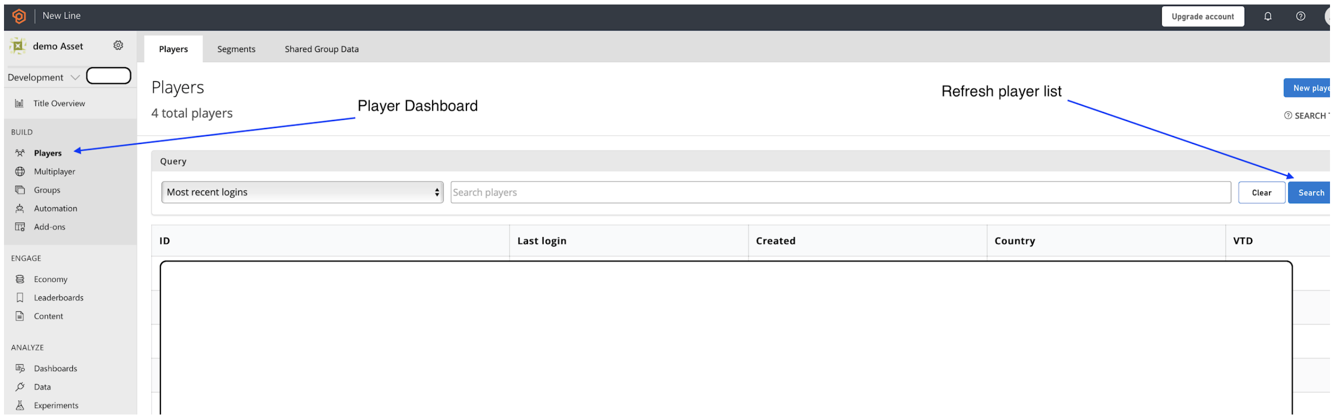
User Id is a string that will contain the id of the currently connected player.

You can find it in the demo script, from lines 59 to 56.

Want more astuces? join us on discord.

How to use PlayFab dashboard

1 - Accessing Users



2 - View his data

Once on the players reference page, click on one of the list to access their profile.

Once on a profile, the Overview page references this personal data, then the Statistics section will show you its data that you have chosen to save online, and finally, the Logins section will show you these latest connections.

The other sections will probably not be useful to you with this asset, as they are not compatible.

To go to your dashboard, in the unity action bar go to: ***Database Controller > Your Dashboard***