

PlayFab Cloud Scripting

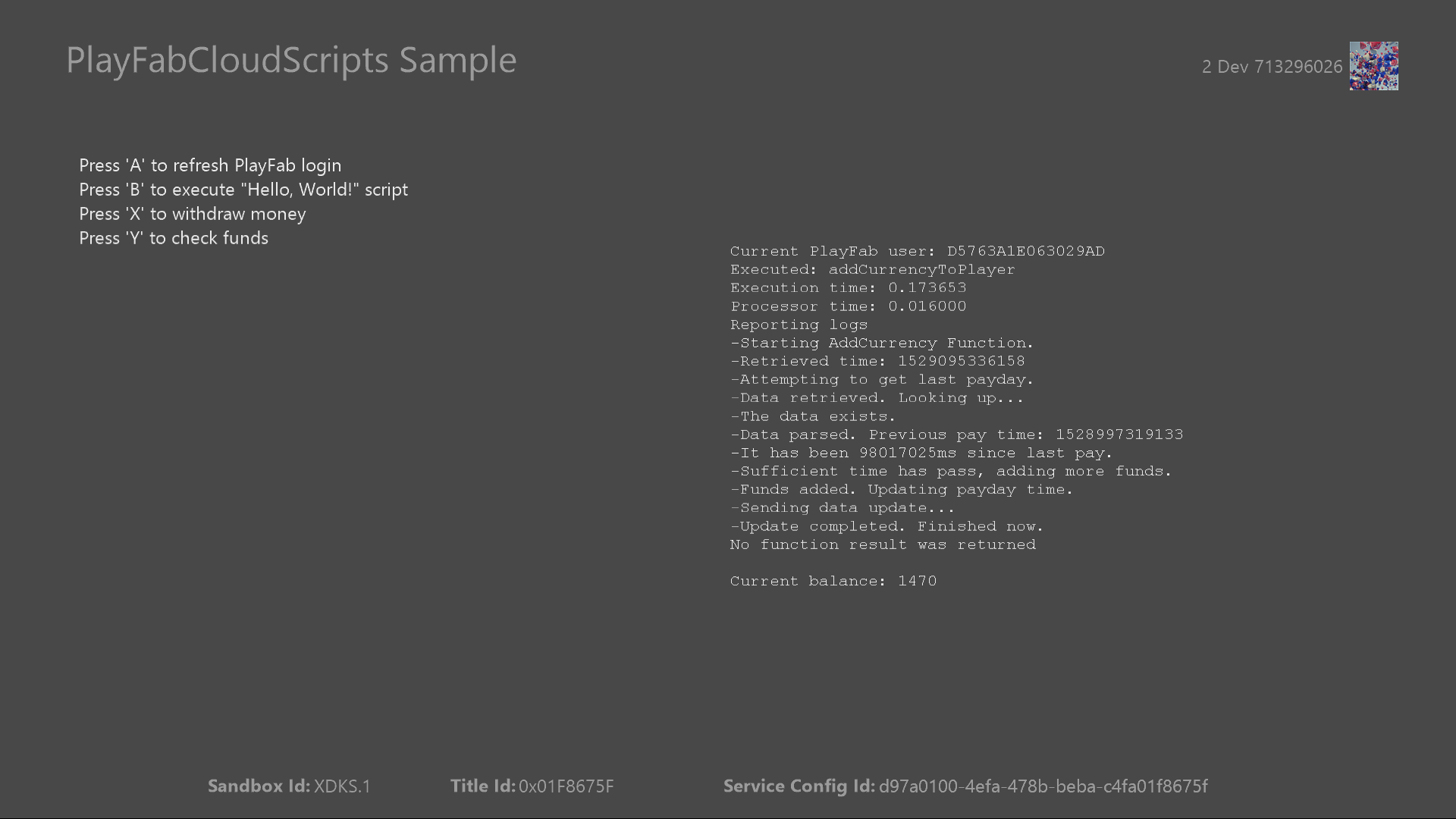
*\* This sample is compatible with the April 2018 XDK*

# Description

This sample demonstrates invoking cloud scripts

# Using the sample

This sample is configured to work in the XDKS.1 sandbox.



|  |  |
| --- | --- |
| Action | Gamepad |
| Refresh PlayFab auth token | A button |
| Run “Hello, World” cloud script | B button |
| Run cloud script to add player funds | X button |
| Query player virtual currency | Y button |
| Open account picker | Menu button |
| Exit | View button |

# Implementation notes

In this version of the XDK, the sample also requires your Xbox One console to use the XDKS.1 **SandboxID**. To switch to this **SandboxID**, in the Xbox One XDK command prompt, use:

xbconfig sandboxid=XDKS.1

The PlayFab title ID must be set in the sample initialization code. This sample uses the PlayFab title ID “DC0” which is assigned to the **PlayFabSettings::titleId** static field.

# Cloud scripting

PlayFab services allow you to create your own custom scripts using JavaScript through your PlayFab dashboard. These scripts can be invoked from your game client using the **ExecuteCloudScript** API or through triggers set up from your dashboard.

These scripts are meant to allow you to securely execute individual or small groups of API calls, such as updating player statistics or granting virtual currency. It is not meant to execute long or continuous scripts in the cloud. There are limits to the cloud scripts which can be run, including their run time, the total size of your scripts, and the number of API calls that can be made from a single script. These limits are based on your subscription level and can be found in the **Limits** section of the **Settings** tab in your dashboard.

This sample ties network activity directly to player input. There are code guards included to prevent players from making requests too frequently. This is not required for any specific XR, but it is a good practice to prevent saturating the network more than necessary.

# “Hello, World” cloud script

PlayFab provides a number of scripts to use as reference. It can be seen in your PlayFab dashboard like this:

// This is a Cloud Script function. "args" is set to the value of the "FunctionParameter"

// parameter of the ExecuteCloudScript API.

// (https://api.playfab.com/Documentation/Client/method/ExecuteCloudScript)

// "context" contains additional information when the Cloud Script function is called from a PlayStream action.

handlers.helloWorld = function (args, context) {

// The pre-defined "currentPlayerId" variable is initialized to the PlayFab ID of the player logged-in on the game client.

// Cloud Script handles authenticating the player automatically.

var message = "Hello " + currentPlayerId + "!";

// You can use the "log" object to write out debugging statements. It has

// three functions corresponding to logging level: debug, info, and error. These functions

// take a message string and an optional object.

log.info(message);

var inputValue = null;

if (args && args.inputValue)

inputValue = args.inputValue;

log.debug("helloWorld:", { input: args.inputValue });

// The value you return from a Cloud Script function is passed back

// to the game client in the ExecuteCloudScript API response, along with any log statements

// and additional diagnostic information, such as any errors returned by API calls or external HTTP

// requests. They are also included in the optional player\_executed\_cloudscript PlayStream event

// generated by the function execution.

// (https://api.playfab.com/playstream/docs/PlayStreamEventModels/player/player\_executed\_cloudscript)

return { messageValue: message };

};

This demonstrates simply passing data between a game client and a cloud script. This script is heavily commented and explains passing arguments to a function as well as the static log object. Once a script has run to completion the **ExecuteCloudScriptResult** object receives the contents of the log as well as the return value of the function and/or any error information if an error occurred in the script itself.

# Making API calls in a cloud script

Relying heavily on the client to perform actions like updating statistics and handling transactions adds surface area where your game is open to malicious attacks. This sample demonstrates how you can reduce the chances of attack by doing these operations in the **addCurrencyToPlayer** cloud script. The body of this script is as follows:

handlers.addCurrencyToPlayer = function (args, context) {

log.debug("Starting script to add currency.");

var timeNow = Date.now();

log.debug("Current server time: " + timeNow);

var getRequest = {

PlayFabId: currentPlayerId, Keys: [ "lastPayDay" ] };

var getResult = server.GetUserInternalData(getRequest);

var lastPayTime = getResult.Data["lastPayDay"];

if(lastPayTime)

{

var lastTimeInMs = lastPayTime.Value;

var timeLapse = timeNow - lastTimeInMs;

log.debug("Time of previous payout: " + lastTimeInMs);

if(timeLapse > 10000)

{

log.debug("Sufficient time has pass, adding more funds.");

var currencyRequest = {

Amount: 10, PlayFabId: currentPlayerId, VirtualCurrency: "DU"

};

server.AddUserVirtualCurrency(currencyRequest);

var updateRequest = {

PlayFabId: currentPlayerId, Data: {

"lastPayDay": timeNow

}};

server.UpdateUserInternalData(updateRequest);

}

else

{

log.debug("Aborting! Funds can only be added once every 10 seconds.");

}

}

else

{

log.debug("This is the first time money is being requested.");

var currencyRequest = {

Amount: 50, PlayFabId: currentPlayerId, VirtualCurrency: "DU"

};

server.AddUserVirtualCurrency(currencyRequest);

var updateRequest = {

PlayFabId: currentPlayerId, Data: {

"lastPayDay": timeNow

}};

server.UpdateUserInternalData(updateRequest);

}

log.debug("Script completed.");

};

This script can be run periodically to add virtual currency to a player’s inventory. It tracks how frequently requests are made using **GetUserInternalData** and **UpdateUserInternalData** which can only be accessed from admin or server code; it cannot be accessed directly from a game client. This prevents a client from spoofing requests to add virtual currency more frequently than is desired. In this case, the server only allows requests as frequently as once every 10 seconds to succeed.

# Triggering cloud scripts based on predefined events

Along with invoking cloud scripts from the client, you can also define scripts that run when certain events occur. These events are set up on the **Rules** page of the **Automation** tab of your PlayFab dashboard. For this sample a simple script has been set up to run every time a player logs in in the title. The script is as follows:

handlers.incrementLoginStat = function (args, context) {

var updateRequest = {

PlayFabId: currentPlayerId, Statistics: [{

StatisticName: "LoginCount", Value: 1

}]};

server.UpdatePlayerStatistics(updateRequest);

};

The rule has been given the triggering event **com.playfab.player\_logged\_in**. An **Action** of type **Execute Cloud Script** has been added to execute **incrementLoginStat** every time the event occurs. You can optionally provide arguments to the cloud script.

# Update history

**Initial Release:** June 2018

# Privacy Statement

When compiling and running a sample, the file name of the sample executable will be sent to Microsoft to help track sample usage. To opt-out of this data collection, you can remove the block of code in Main.cpp labeled “Sample Usage Telemetry”.

For more information about Microsoft’s privacy policies in general, see the [Microsoft Privacy Statement](https://privacy.microsoft.com/en-us/privacystatement/).