

PlayFab 云脚本

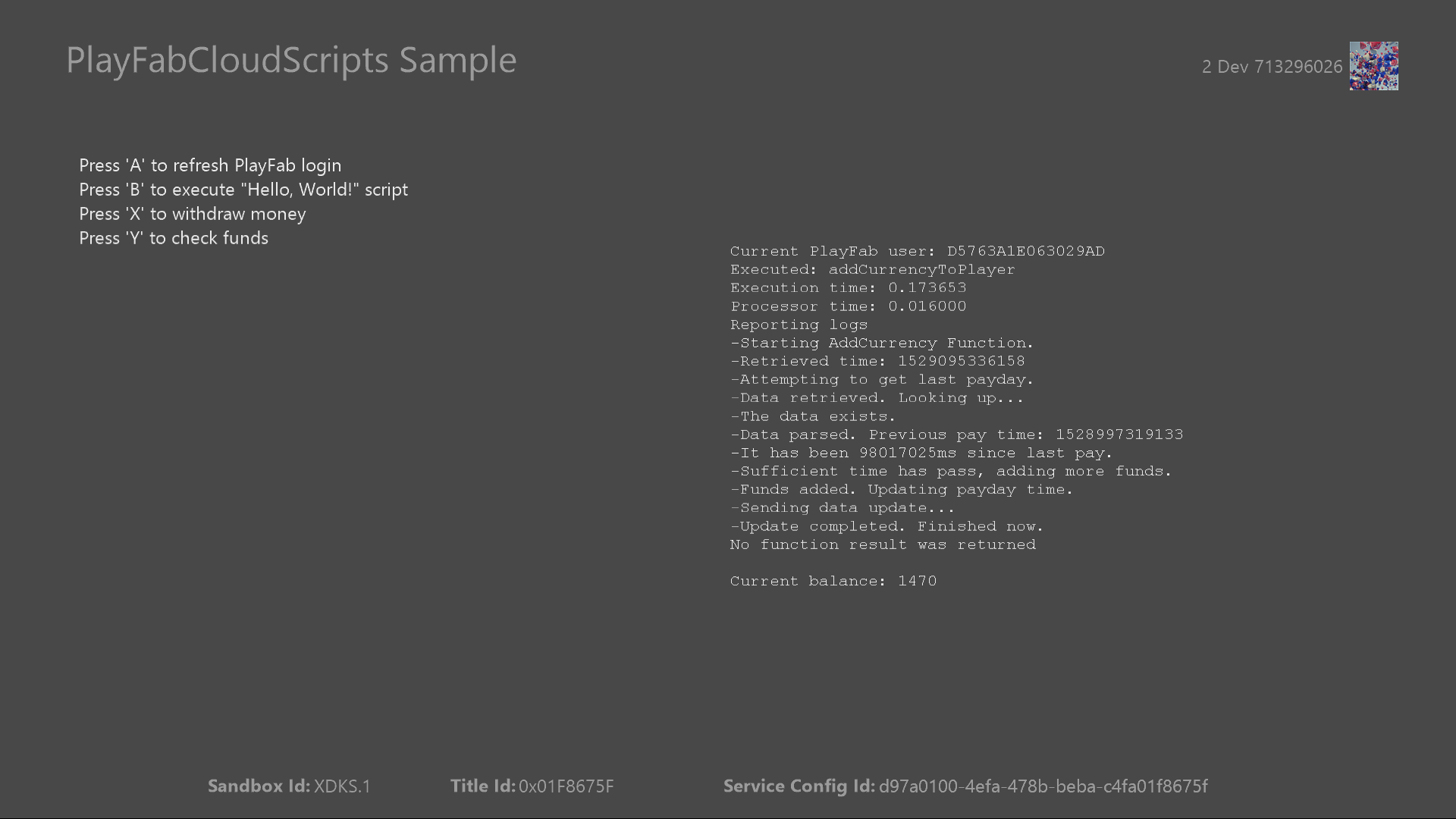
*\*此样本与2018年4月的XDK兼容*

# 描述

此示例演示了调用云脚本

# 使用样本

此示例配置为在XDKS.1沙箱中工作。



|  |  |
| --- | --- |
| 操作 | 游戏手柄 |
| 刷新 PlayFab 身份验证令牌 | A 键 |
| 运行“Hello, World”云脚本 | B 键 |
| 运行云脚本以为玩家充值 | X 键 |
| 查询玩家虚拟货币 | Y 键 |
| 打开帐户选取器 | 菜单按键 |
| 退出 | 查看按键 |

# 实施说明

在此版本的XDK中，该示例还要求您的Xbox One控制台使用XDKS.1 **SandboxID**。要切换到这个 **SandboxID**，在Xbox One XDK命令提示符下，使用：

xbconfig sandboxid = XDKS.1

必须在样本初始化代码中设置PlayFab标题ID。此示例使用分配给的PlayFab标题ID“DC0” **PlayFabSettings::titleId** 静态场。

# 云脚本

PlayFab 服务允许您通过 PlayFab 仪表板使用 JavaScript 创建自己的自定义脚本。可以使用 **ExecuteCloudScript** API 或通过从仪表板设置的触发器，从您的游戏客户端调用这些脚本。

这些脚本旨在允许您安全执行单个或小组 API 调用，例如，更新玩家统计信息或授予虚拟货币。它并不非在云中执行长脚本或连续脚本。可以运行的云脚本有限制，包括脚本运行时间、脚本的总大小以及可以从单个脚本进行的 API 调用的数量。这些限制基于您的订阅级别，可以在仪表板**设置**选项卡的**限制**部分找到。

此示例将网络活动直接与玩家输入相联系。已包含一些代码保护，以防止玩家过于频繁地提出请求。这对于任何特定的 XR 都不是必需的，但对防止网络不必要的饱和却是一个不错的方法。

# “Hello, World”云脚本

PlayFab 提供了一些脚本作为参考。PlayFab 仪表板内容如下：

// 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 };

};

这仅仅演示了在游戏客户端和云脚本之间传递数据。此脚本有很多注释，并解释了将参数传递到函数以及静态日志对象的过程。脚本运行完成 **ExecuteCloudScriptResult** 后，如果脚本本身发生错误，则对象将接收日志的内容以及函数的返回值和/或任何错误信息。

# 在云脚本中进行 API 调用

如果在很大程度上依赖客户端执行更新统计信息和处理交易等操作，则会增加游戏对恶意攻击开放的外围应用。此示例演示了如何通过在 **addCurrencyToPlayer** 云脚本中执行这些操作来减少攻击的可能性。此脚本的主体如下：

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.");

};

可以定期运行此脚本，以将虚拟货币添加到玩家的库存中。它将跟踪使用 **GetUserInternalData** 和 **UpdateUserInternalData**（只能从管理员或服务器代码访问，不能直接从游戏客户端访问）提出请求的频率。这可以防止客户端出现比预期更频繁地添加虚拟货币等欺诈请求。在这种情况下，服务器仅允许请求每 10 秒成功一次。

# 基于预定义事件触发云脚本

除了从客户端调用云脚本外，还可以定义在发生特定事件时运行的脚本。这些事件可在 PlayFab 仪表板**自动化**选项卡的**规则**页面加以设置。对于此示例，已设置一个简单的脚本，以便玩家每次登录标题时运行。脚本如下：

handlers.incrementLoginStat = function (args, context) {

var updateRequest = {

PlayFabId: currentPlayerId, Statistics: [{

StatisticName: "LoginCount", Value: 1

}]};

server.UpdatePlayerStatistics(updateRequest);

};

该规则已被赋予触发事件 **com.playfab.player\_logged\_in**。每次事件发生时，都已添加一个类型为**执行云脚本**的**操作**来执行 **incrementLoginStat**。您可以选择为云脚本提供参数。

# 更新历史记录

**初始发行：**2018 年 6 月

# 隐私声明

编译和运行示例时，示例可执行文件的文件名将发送给Microsoft以帮助跟踪示例使用情况。要选择退出此数据收集，您可以删除Main.cpp中标记为“Sample Usage Telemetry”的代码块。

有关 Microsoft 隐私政策的更多信息，请参阅 [Microsoft 隐私声明](https://privacy.microsoft.com/zh-cn/privacystatement/)。