

PlayFab Web Auth Broker Sample

*\* This sample is compatible with the July 2018 Xbox One XDK*

# Description

On Xbox One, in order to use Xbox Live services the player must authenticate with Xbox Live. In addition to that, if you want your title to use PlayFab services then you will need to create or otherwise acquire a PlayFab account that you can subsequently link with the Xbox Live user account. There are many ways to acquire a PlayFab account. The simplest method is to silently create the PlayFab account the first time the player signs into the game. This “simple” method is the typical sign-in flow for an Xbox One game and is the recommended scenario for most games. (If you are new to PlayFab authentication, then please start with the SimplePlayFabAuth sample). The PlayFabWebAuthBroker sample demonstrates a more advanced sign-in flow, that is appropriate for games that want to enable sharing a PlayFab account from another authentication provider. In particular, this sample demonstrates how to acquire a PlayFab account by performing Open Authentication (OAUTH) with an open authentication provider (in this case, Facebook).

# Building the Sample

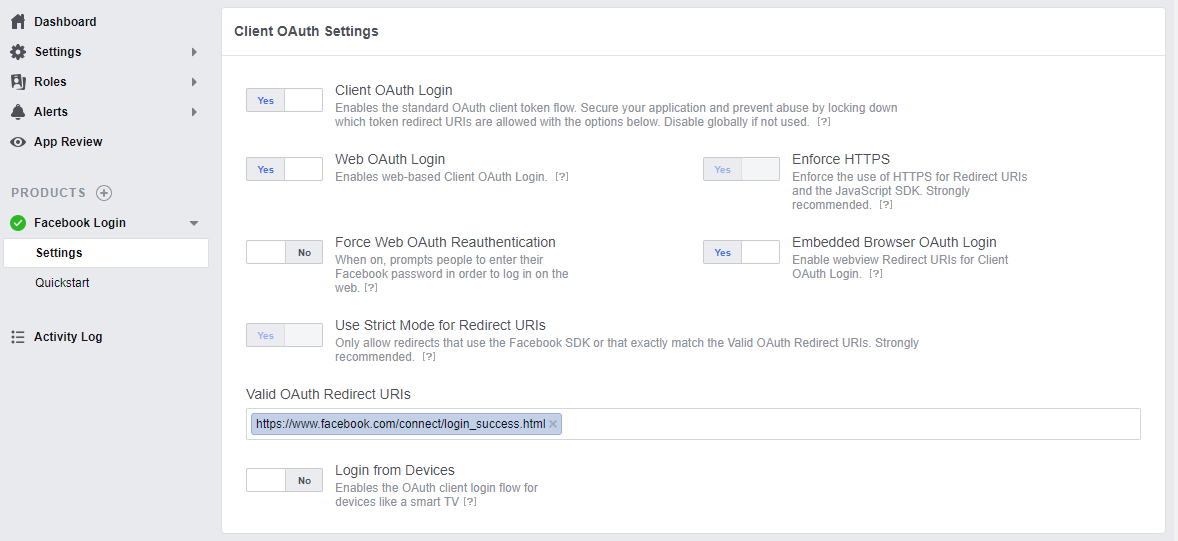
The sample builds with Visual Studio 2015 and has been tested with the July 2018 Xbox One XDK. Note that the sample will likely work with earlier XDKs

# Configuring a Facebook App for the Sample

This sample uses Facebook as an example OAUTH provider. In order to use this sample you will have to set up an application on Facebook. More information about setting up an app on Facebook can be found here <https://developers.facebook.com/docs/apps>.

## Facebook Login Product

When you set up your app you will need to add the Facebook Login Product. Once you’ve added the product, you must specify a valid OAuth redirect URI. For example:



## Test Users

Create one or more Test Users to use when signing in to Facebook from the sample. Test User accounts are already configured to enable your app on Facebook and are much more convenient for experimenting with the sample.

## Setting the APP ID

Before you can use the sample, you will need to edit FacebookAuthHelper.h and provide your Facebook APP ID. Locate the App ID in the Basic Settings for your app on Facebook. Next, look for the “TODO” comment in FacebookAuthHelper.h, and replace “<put your Facebook App ID here>” with your App ID:



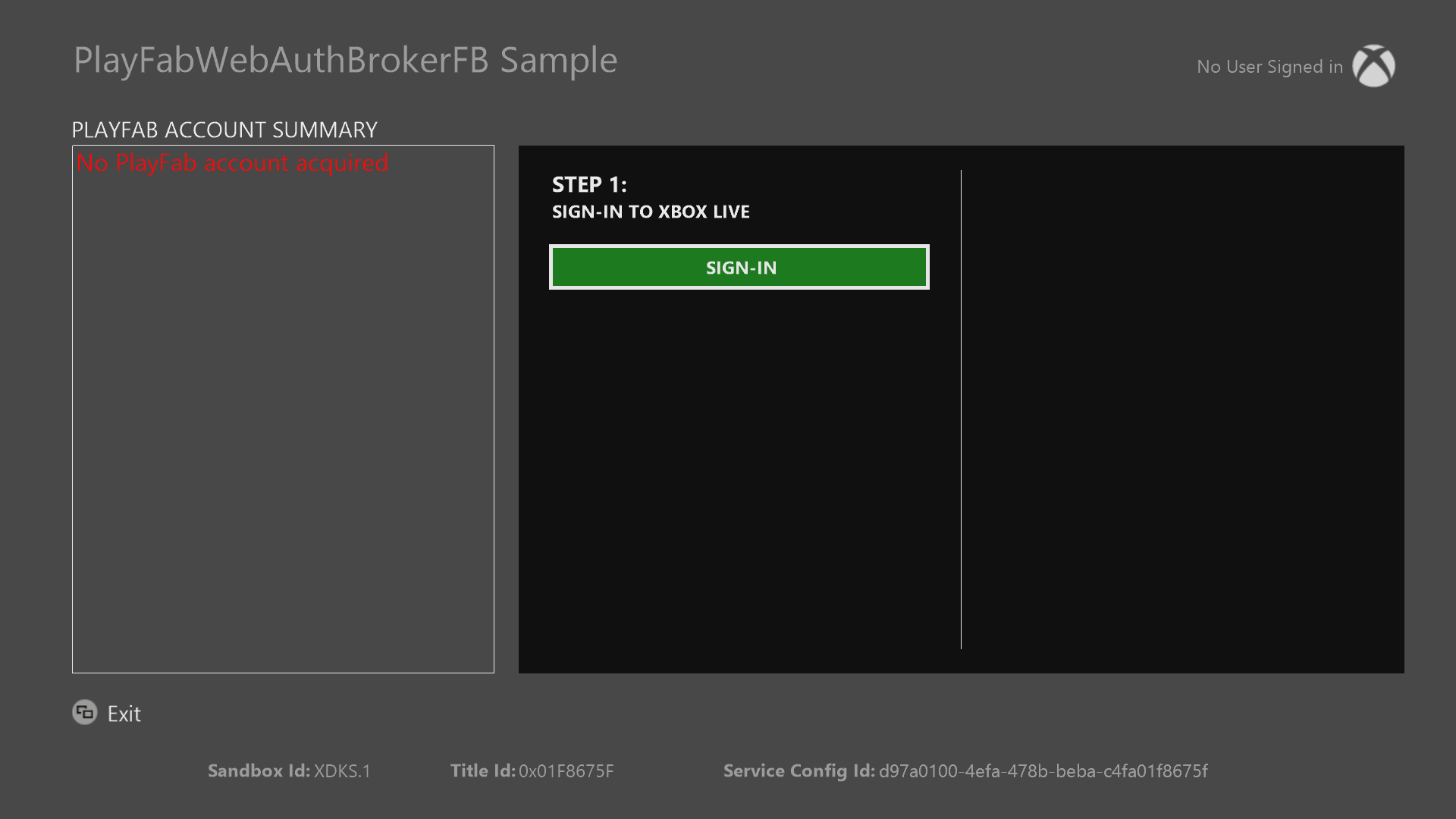
# Using the sample

Once you have everything set up on Facebook, and you have entered your Facebook App ID into FacebookAuthHelper.h then using the sample is straightforward. You will be presented with a simple sign-in flow analogous to what you might see in a game. However, each screen also displays diagnostic information that you would normally not display for the end user. There is also a panel that will display account summary information once the sign-in flow is complete.

## Sign-In Flow

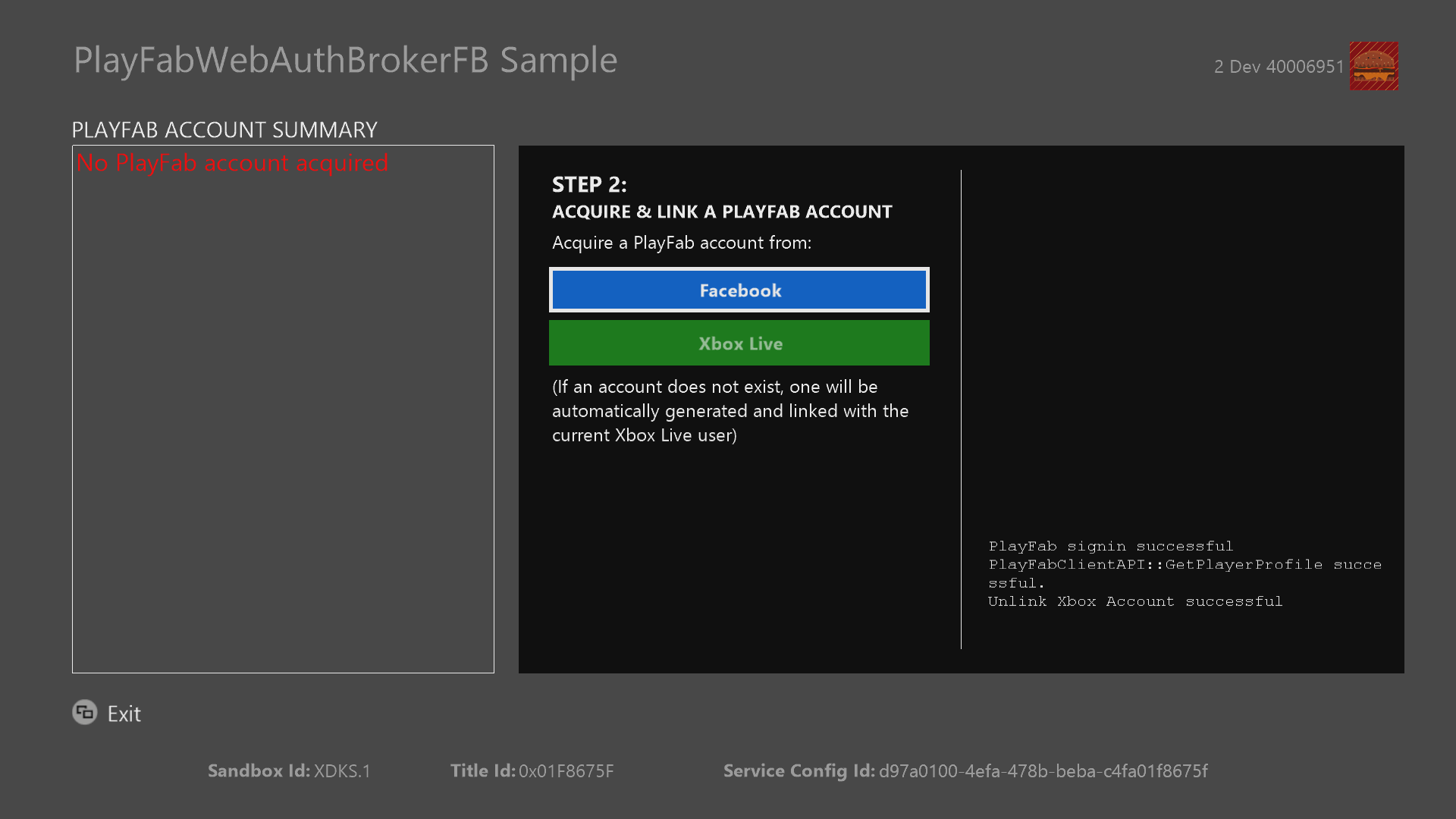
## Xbox Live Sign-In

The first step in the sign-in flow is signing into Xbox Live. The user must always sign into Xbox Live before making service calls for Xbox Live or PlayFab.



## Acquiring a PlayFab account

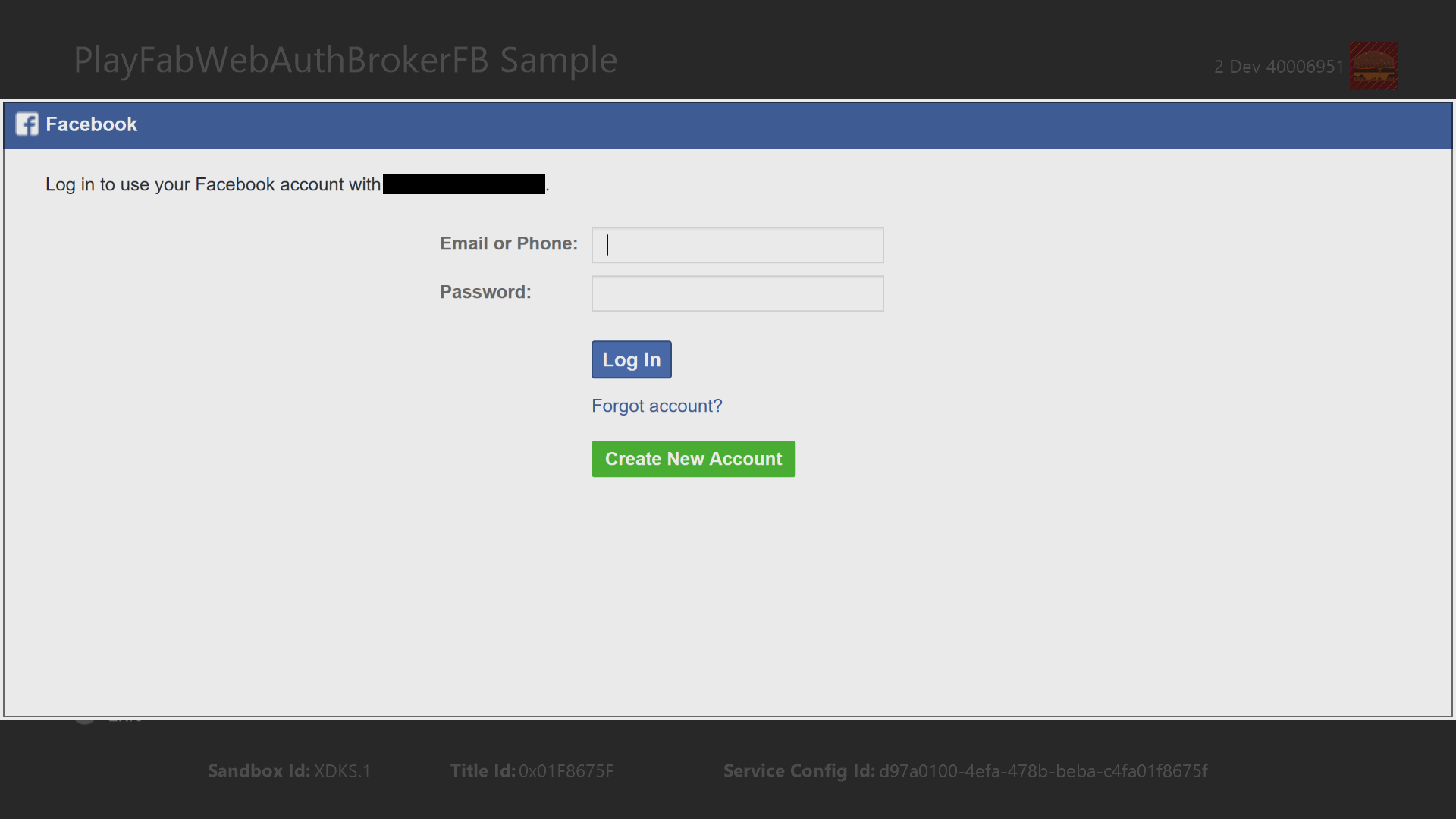
To use PlayFab services you will need to link a PlayFab account with the Xbox Live user. There are many ways to come up with a PlayFab account. In the second step of the sign-in flow the user is presented with two choices. For the “Xbox Live” choice, the sample simply creates a default PlayFab account to link with the Xbox Live user account. For the “Facebook” choice, this sample demonstrates how you can acquire a PlayFab account that is already linked to another login provider, such as Facebook. By performing OAuth with Facebook you can potentially discover a PlayFab account that has already been linked with the player’s Facebook account and then subsequently link this account with the player’s Xbox Live account.



For more information about PlayFab sign-in options please see the PlayFab documentation on [Login Basics & Best Practices](https://api.playfab.com/docs/tutorials/landing-players/best-login)

## Facebook OAuth UI

When you choose to authenticate with Facebook, the sample calls Windows::Xbox::UI::SystemUI::AuthenticateAsync. This API call shows the following web-based system UI:



This is where you will provide the authentication credentials for one of the Test Users that you created for your Facebook App. This UI is rendered by the system and not by the title. Like other system UIs, this window supports using the system keyboard, however, for testing purposes, it is more convenient to use a physical keyboard.

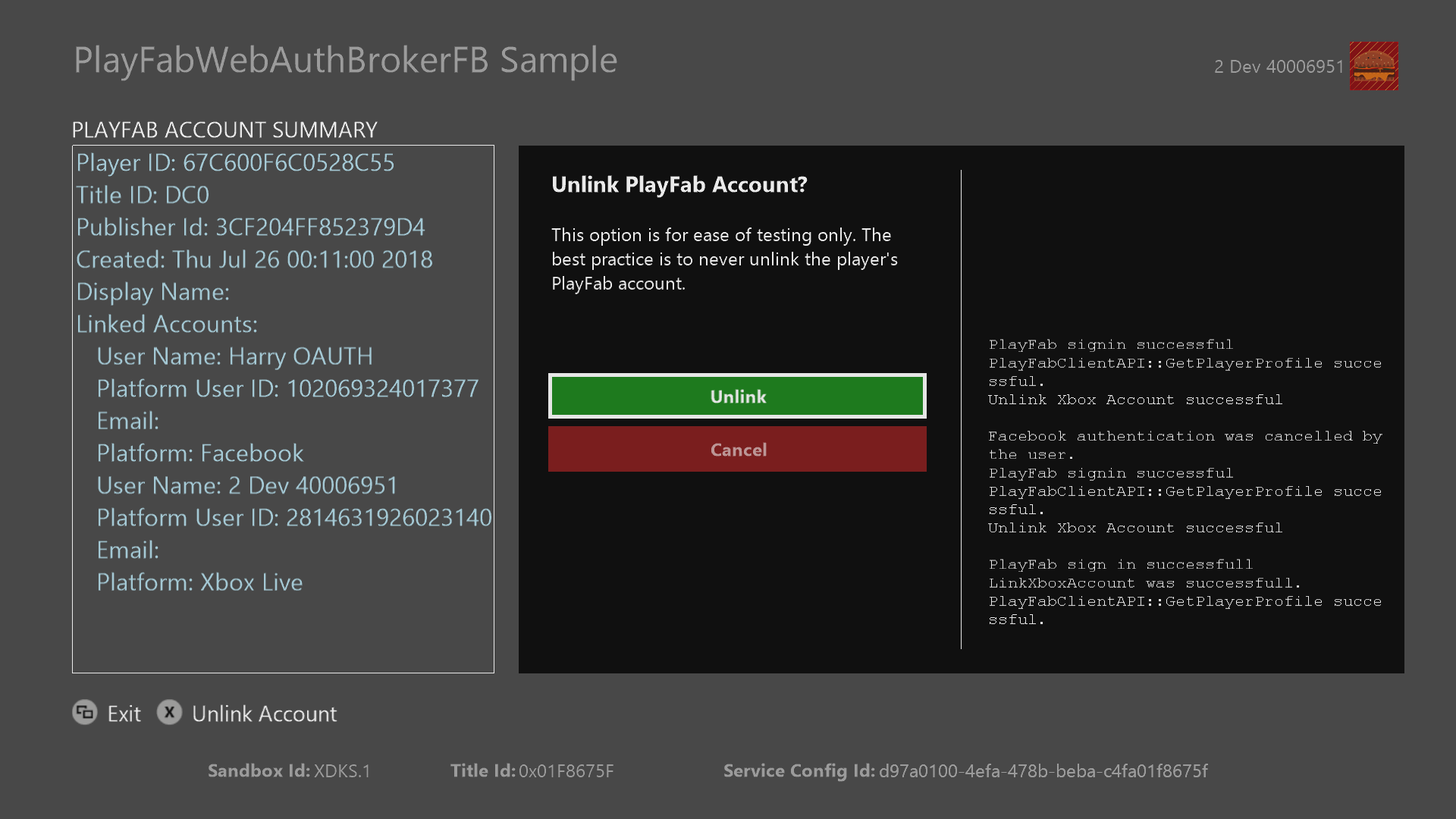
## Sign-In Complete

Once the sign-in flow is complete, the final screen (Step 3) is displayed. This is the point where your game would be ready to start making both Xbox Live and PlayFab service calls. Note that the sample displays PlayFab account summary information on the left-hand side of the screen. This information is provided via a call to PlayFabClientAPI::GetPlayerProfile.

## 

## Unlinking the PlayFab account from the Xbox Live account

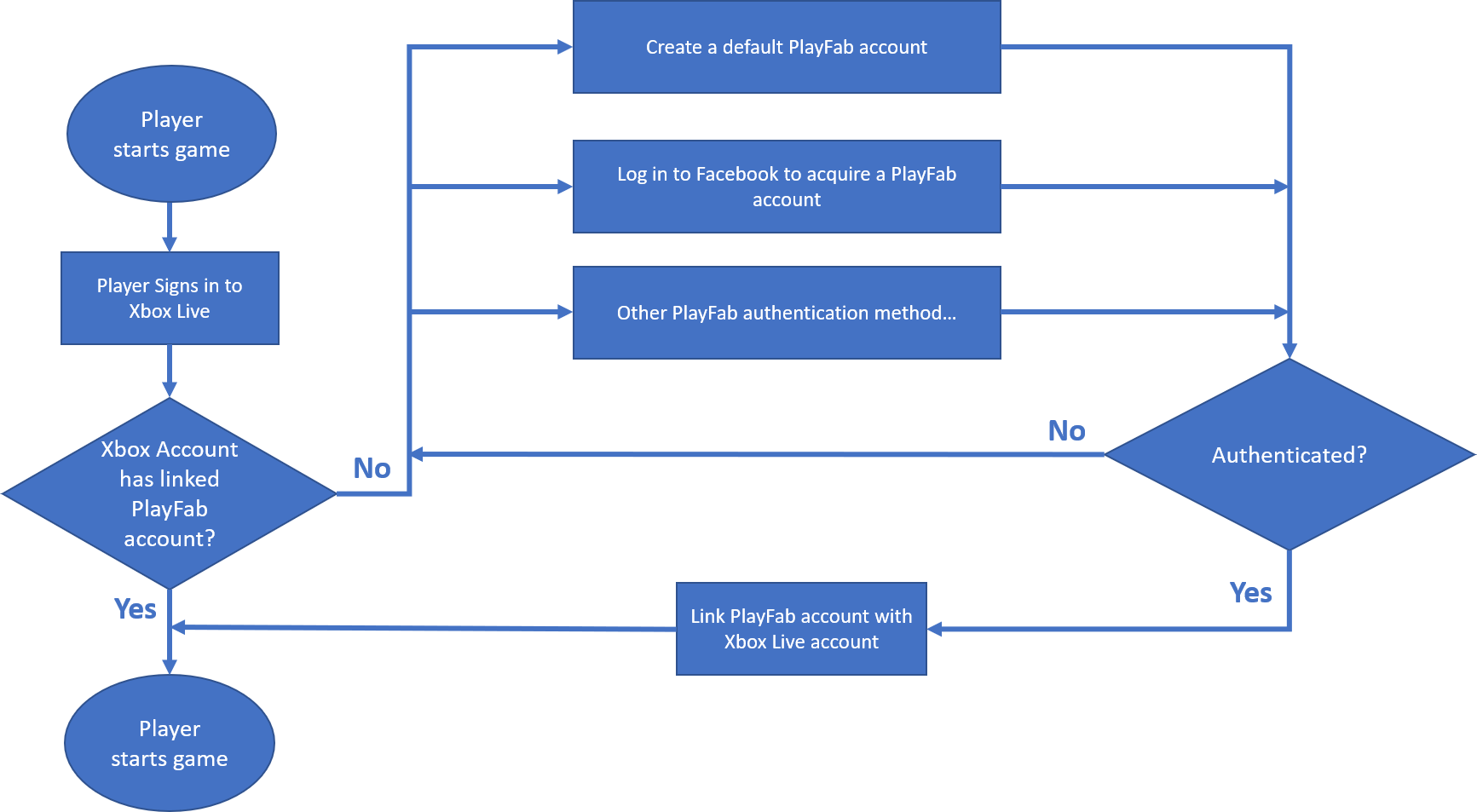
Once you have successfully completed the sign-in flow, the sample will provide the option to unlink the PlayFab account from the Xbox Live account. The PlayFab SDK provides an API (PlayFabClientAPI::UnlinkXboxAccount) for this purpose however, as a best practice, you would normally never unlink a player’s PlayFab account. When you unlink an account, any PlayFab server-side state that is maintained for the player could potentially be lost. Furthermore, same account linking could very well be shared between multiple-titles that share the same Publisher ID for PlayFab. So the consequences of unlinking the account could be broad, destructive, and confusing for players. For this reason, the sample only provides the Unlink functionality as a convenience for testing purposes.



# Implementation notes

## Advanced Sign-In Flow

The sample implements the following sign-in flow:



This is flowchart is a simplified version of the flow chart from the PlayFab documentation on [Login Basics & Best Practices](https://api.playfab.com/docs/tutorials/landing-players/best-login) The sign-in flow implemented by the sample is also more appropriate for Xbox One titles because it always assumes that the player will sign-in with Xbox Live as the first step. The logical sign-in flow is implemented in PlayFabWebAuthBrokerFB.h/.cpp.

**Linking PlayFab Accounts**

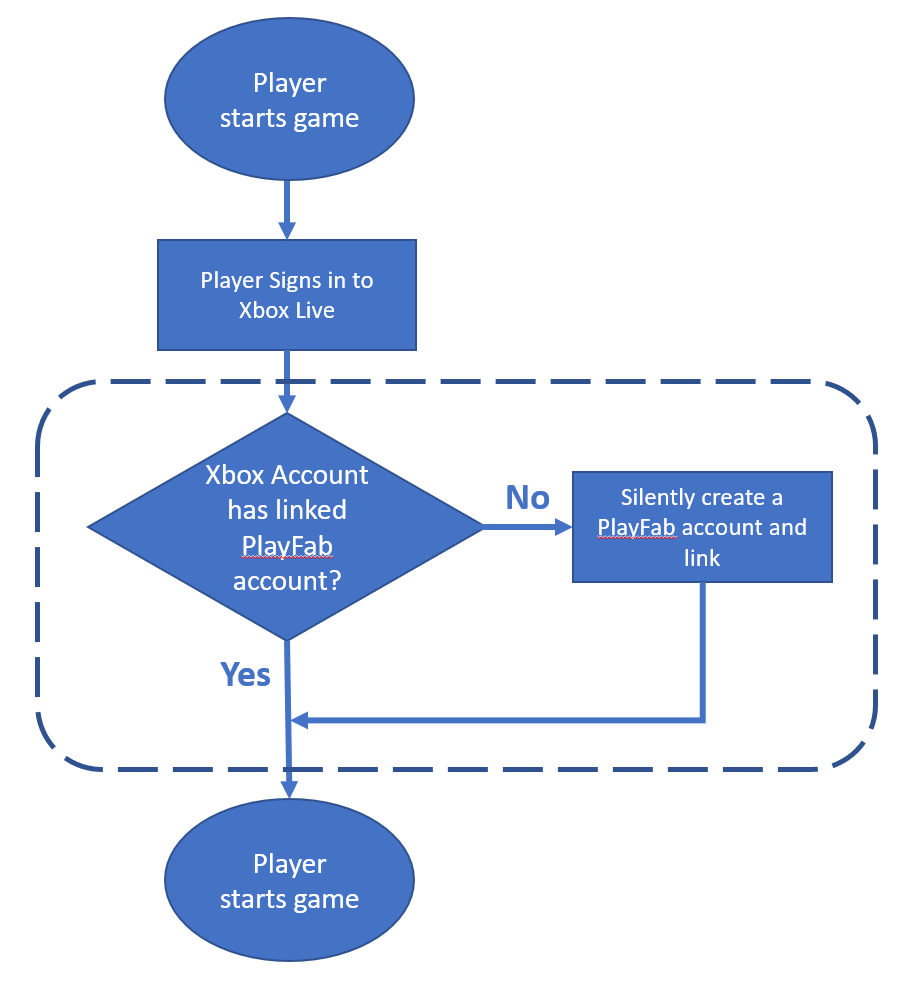
When implementing the advanced sign-in flow, you need to be careful whenever you link a PlayfFab account with an Xbox Live account. If the Xbox Live account is already linked to a PlayFab account, then when you “force link” a different PlayFab account, the original account will be abandoned. This can be confusing and frustrating for the player if they lose progress or other account state associated with the original PlayFab account. The sign-in flow shown here avoids this issue because it tests for the existence of a PlayFab account before presenting any other choices to the player. Specifically:

* In the error handler for PlayFabClientAPI::LoginWithXbox, the sample checks for PlayFabErrorCode::PlayFabErrorAccountNotFound. The presence of this error informs the sample’s logic that there is no PlayFab account already linked with the Xbox Live account
* In the PlayFabClientAPI::LinkXboxAccountRequest, the sample sets the ForceLink option to ***false***. If you follow the recommended sign-in flow then you should never have a reason to set this option to true.

The best practice is to *always use caution when setting ForceLink to true* to avoid unnecessarily abandoning an existing PlayFab account.

**Simple Sign-In flow**

The following diagram shows a minimal sign-in flow.



Although simplified sign-in flow is minimal, it is still appropriate for most Xbox One games that rely on Xbox Live as the sole authentication provider. Note that everything within the dashed lines can be accomplished *silently* without presenting the player with additional UI flow. In particular, when you fill out the LoginWithXboxRequest structure, specify CreateAccount = true to indicate that a new PlayFab account should be created in the case when an account does not already exist. For most games, this will be the preferred way to go.

## OAuth Implementation

The Facebook OAuth implementation is provided in a stand-alone class, called FacebookAuthHelper. This is a simplified version of the “FacebookHelper” that is part of another sample called “WebAuthBrokerFB”. If you want to perform OAuth against an authentication provider different from Facebook then you can start with FacebookAuthHelper as a preliminary example and modify it as appropriate.

# Known issues

## Windows::Xbox::UI::SystemUI::AuthenticateAsync

* Sometimes the dialog becomes unresponsive and won't accept physical keyboard input. Work-around by restarting the sample
* After successfully logging in, subsequent calls may not show the dialog but rather the async operation will complete successfully, perhaps using cached login results from the previous successful login.
* The API throws an exception after every successful login. The exception is handled and can be ignored but is annoying for developers.

# Update history

First release of sample November 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/).