Sign in with Apple Unity Plugin

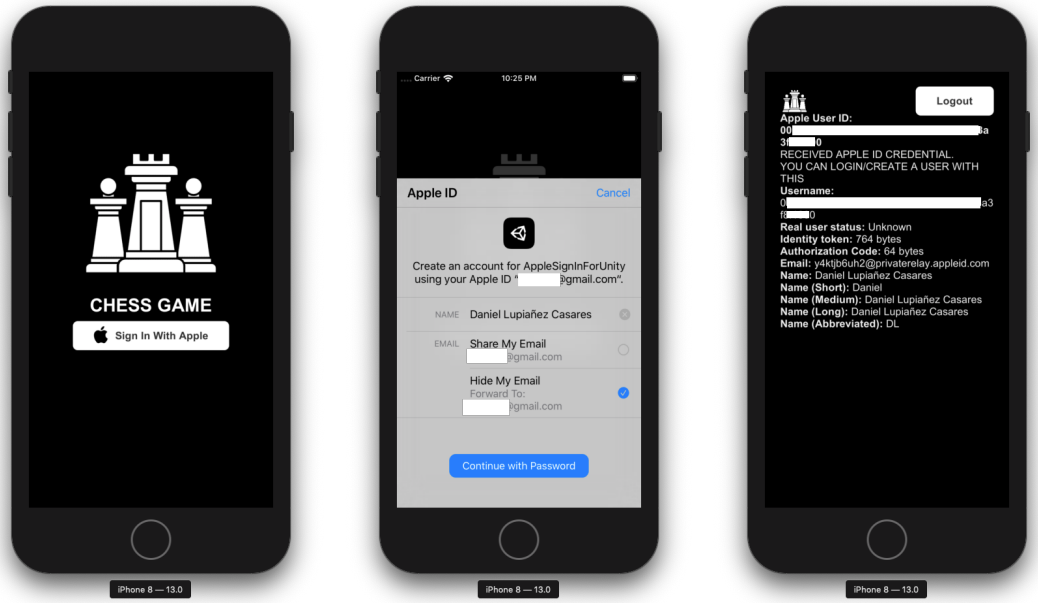
by **Daniel Lupiañez Casares**

release v1.4.0 license MIT CHANGELOG

openupm v1.4.0

Stars 176 Followers 50

Follow @lupi_dan 252



Overview

Sign in with Apple plugin to use with Unity 3D game engine.

This plugin supports the following platforms: * **iOS** * **macOS** (NOTES) * **tvOS** (Experimental)

The main purpose for this plugin is to expose Apple's newest feature, Sign in with Apple, to the Unity game engine.

On WWDC19, Apple announced **Sign in with Apple**, and on top of that, they announced that every iOS/tvOS/macOS Application that used any kind of Third party sign-ins (like *Sign in with Facebook*, or *Sign in with Google*), will have to support Sign in with Apple in order to get approved for the App Store, making it **mandatory**.

Features

Native Sign in with Apple

- Support for iOS
- Support for macOS (NOTES)
- Support for tvOS (Experimental)
- Supports Sign in with Apple, with customizable scopes (Email and Full name).
- Supports Get Credential status (Authorized, Revoked and Not Found).
- Supports Quick login (including iTunes Keychain credentials).
- Supports adding Sign In with Apple capability to Xcode project programatically in a PostBuild script.
- Supports listening to Credentials Revoked notifications.
- Supports setting custom Nonce and State for authorization requests when Signing In, and attempting a Quick Login.
- NSError mapping so no details are missing.
- NSPersonNameComponents support (for ALL different styles).
- Customizable serialization (uses Unity default serialization, but you can add your own implementation)

Installation

> Current stable version is v1.4.0

There are two options available to install this plugin. Either using the Unity Package Manager, or the traditional `.unitypackage` file.

Unity Package Manager

Install via Git URL

Available starting from Unity 2018.3.

Just add this line to the `Packages/manifest.json` file of your Unity Project:

```
"dependencies": {
    "com.lupidan.apple-signin-unity": "https://github.com/lupidan/apple-signin-unity.git#v1.4.0",
}
```

Install via OpenUPM

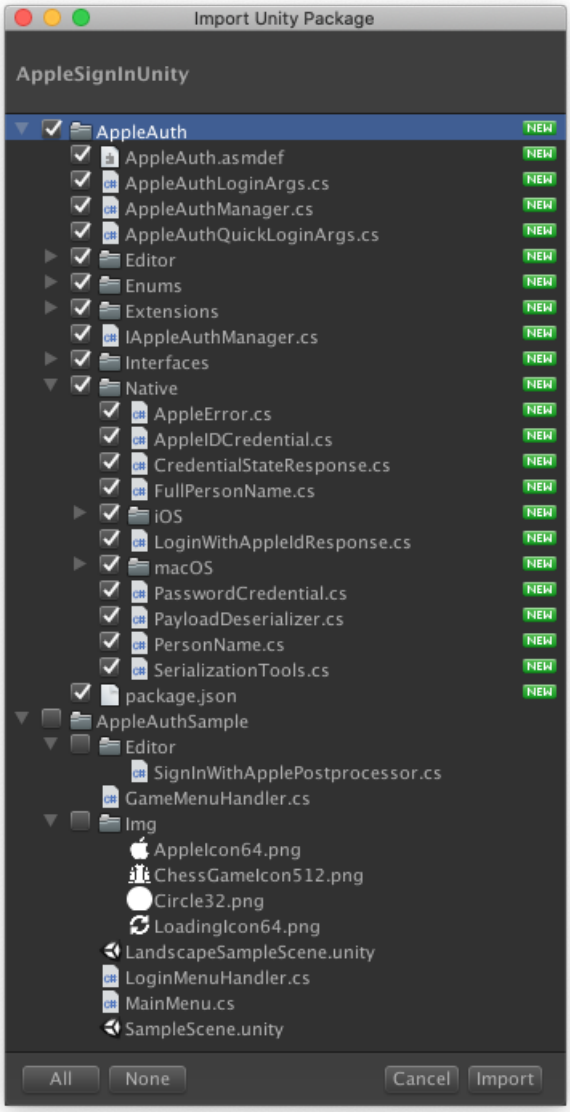
The package is available on the openupm registry. You can install it via openupm-cli.

```
openupm add com.lupidan.apple-signin-unity
```

Unity Package File

1. Download the most recent Unity package release here
2. Import the downloaded Unity package in your app. There are two main folders:

- The `AppleAuth` folder contains the **main plugin**.
- The `AppleAuthSample` folder contains **sample code** to use as a reference, or to test the plugin.



Plugin setup (iOS/tvOS)

To be able to use Apple's platform and framework for Authenticating with an Apple ID, we need to set up our Xcode project. Two different options are available to set up the entitlements required to enable Apple ID authentication with the iOS SDK.

Programmatic setup with a Script

RECOMMENDED

This plugin **provides an extension method** for `ProjectCapabilityManager` (docs), used to add this entitlement programatically after an Xcode build has finished.

Simply create a Post Processing build script (more info) that performs the call. If you already have a post process build script, it should be simple to add to your code.

The provided extension method is `AddSignInWithAppleWithCompatibility`. It accepts an optional argument for Unity 2019.3 to indicate the UnityFramework target Guid.

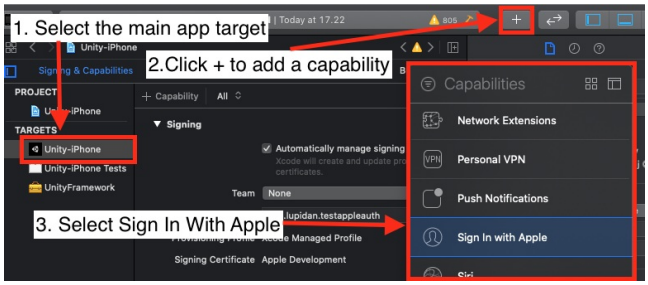
Sample code:

```
using AppleAuth.Editor;

public static class SignInWithApplePostprocessor
{
    [PostProcessBuild(1)]
    public static void OnPostProcessBuild(BuildTarget target, string path)
    {
        if (target != BuildTarget.iOS)
            return;

        var projectPath = PBXProject.GetPBXProjectPath(path);

        // Adds entitlement depending on the Unity version used
#if UNITY_2019_3_OR_NEWER
            var project = new PBXProject();
            project.ReadFromString(System.IO.File.ReadAllText(projectPath));
            var manager = new ProjectCapabilityManager(projectPath, "Entitlements.entitlements", null, project.GetUnityMainTargetGui
            manager.AddSignInWithAppleWithCompatibility(project.GetUnityFrameworkTargetGuid());
            manager.WriteToFile();
#else
            var manager = new ProjectCapabilityManager(projectPath, "Entitlements.entitlements", PBXProject.GetUnityTargetName());
            manager.AddSignInWithAppleWithCompatibility();
            manager.WriteToFile();
#endif
    }
}
```

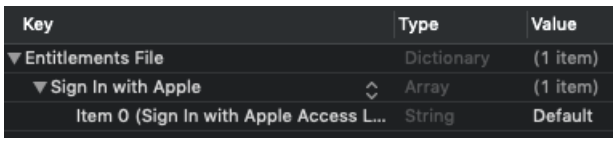
Manual entitlements setup

The other option is to manually setup all the entitlements in our Xcode project. Note that when making an iOS Build from Unity into the same folder, if you choose the option to overwrite, you will need to perform the Manual setup again.

1. In your generated Xcode project. Select the main app Unity-iPhone target and select the option *Signing And Capabilities*. You should see there an option to add a capability from a list. Just locate *Sign In With Apple* and add it to your project.



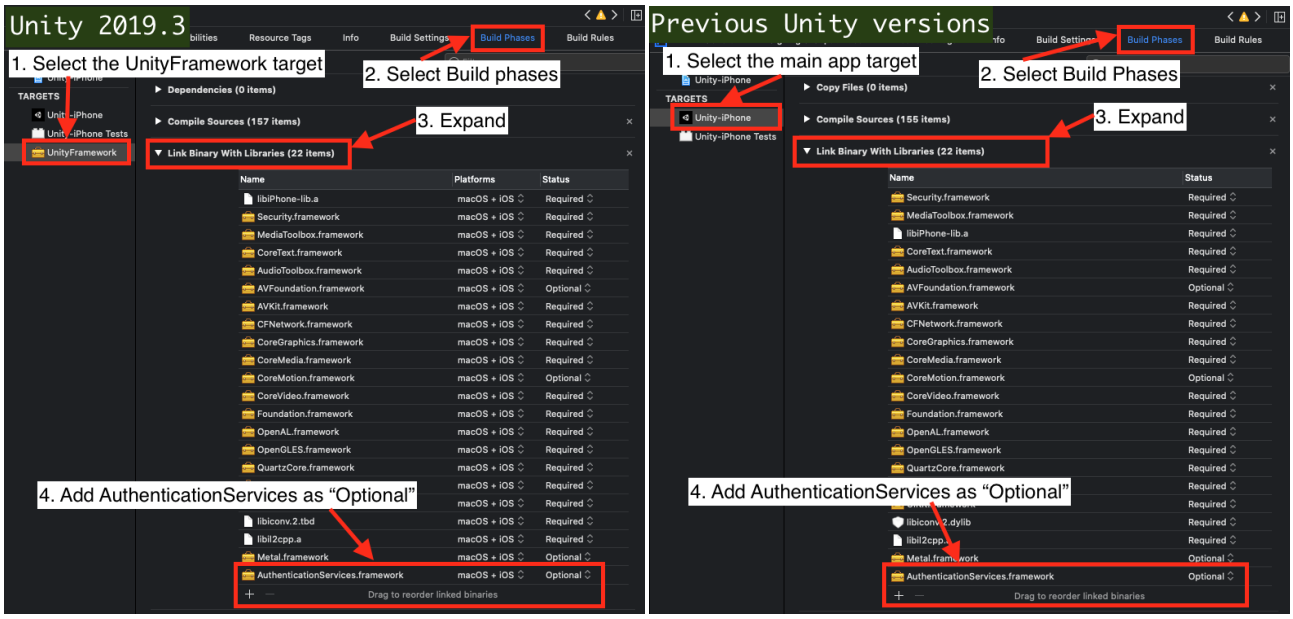
2. This should have added an Entitlements file to your project. Locate it on the project explorer (it should be a file with the extension `.entitlements`). Inside it you should see an entry like this one:



3. You need to import the `AuthenticationServices.framework` library in the Build Phases->Link Binary with Libraries. **If you are targeting older iOS versions**, mark the library as `Optional`.

   For **Unity 2019.3** onwards, add it to the UnityFramework target

   For **previous Unity versions**, add it to the main Unity-iPhone target



Enabling Apple capability

You will also need to **setup everything** in the Apple's developer portal. More information can be found here. Please remember this plugin only supports native Sign In With Apple on iOS (no REST API support).

There is also a Getting Started site.

Final notes regarding setup

The `AuthenticationServices.framework` should be added as Optional, to support previous iOS versions, avoiding crashes at startup.

The provided extension method uses reflection to integrate with the current tools Unity provides. It has been tested with Unity 2018.x and 2019.x. But if it fails on your particular Unity version, feel free to open a issue, specifying the Unity version.

Plugin setup (macOS)

An unsigned precompiled `.bundle` file is available. It will be automatically included in your macOS builds. However that `.bundle` needs to be modified to avoid issues when uploading it to the MacOS App Store.

In particular, the bundle identifier of that `.bundle` needs to be modified to a custom one.

To automate the process, there is a helper method that will change the bundle identifier to one based on your project's application identifier. You should call this method on a Postprocess build script of your choice.

```
using AppleAuth.Editor;

public static class SignInWithApplePostprocessor
{
    [PostProcessBuild(1)]
    public static void OnPostProcessBuild(BuildTarget target, string path)
    {
        if (target != BuildTarget.StandaloneOSX)
            return;

        AppleAuthMacosPostprocessorHelper.FixManagerBundleIdentifier(target, path);
    }
}
```

The Xcode project with the source code to generate a new bundle file is available at `MacOSAppleAuthManager/MacOSAppleAuthManager.xcodeproj`

To support the feature, **the app needs to be codesigned correctly**, including the required entitlements. For more information regarding macOS codesign, please follow this link.

Implement Sign in With Apple

> Currently, it seems Sign In With Apple does not work properly in the simulator. This needs testing on a device with an iOS 13 version.

An overall flow of how the native Sign In With Apple flow works is presented in this diagram. There is no official documentation about it, the only available source for this is the WWDC 2019 talk. You can watch it here

Initializing

```
private IAppleAuthManager appleAuthManager;

void Start()
{
    ...
   // If the current platform is supported
   if (AppleAuthManager.IsCurrentPlatformSupported)
   {
       // Creates a default JSON deserializer, to transform JSON Native responses to C# instances
       var deserializer = new PayloadDeserializer();
       // Creates an Apple Authentication manager with the deserializer
       this.appleAuthManager = new AppleAuthManager(deserializer);
   }
    ...
}

void Update()
{
    ...
    // Updates the AppleAuthManager instance to execute
    // pending callbacks inside Unity's execution loop
    if (this.appleAuthManager != null)
    {
        this.appleAuthManager.Update();
    }
    ...
}
```

Perform Sign In With Apple

If you want to Sign In and request the Email and Full Name for a user, you can do it like this:

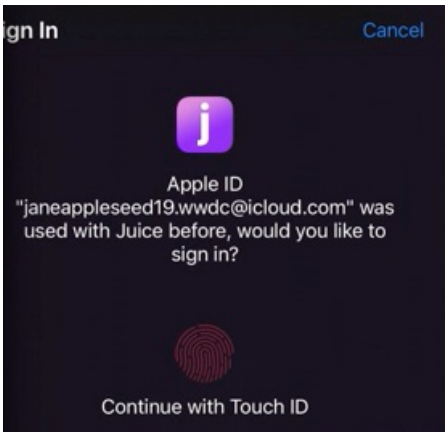
```
var loginArgs = new AppleAuthLoginArgs(LoginOptions.IncludeEmail | LoginOptions.IncludeFullName);

this.appleAuthManager.LoginWithAppleId(
    loginArgs,
    credential =>
    {
        // Obtained credential, cast it to IAppleIDCredential
        var appleIdCredential = credential as IAppleIDCredential;
        if (appleIdCredential != null)
        {
            // Apple User ID
            // You should save the user ID somewhere in the device
            var userId = appleIdCredential.User;
            PlayerPrefs.SetString(AppleUserIdKey, userId);

            // Email (Received ONLY in the first login)
            var email = appleIdCredential.Email;

            // Full name (Received ONLY in the first login)
            var fullName = appleIdCredential.FullName

            // Identity token
            var identityToken = Encoding.UTF8.GetString(
                appleIdCredential.IdentityToken,
                0,
                appleIdCredential.IdentityToken.Length);

            // Authorization code
            var authorizationCode = Encoding.UTF8.GetString(
                appleIdCredential.AuthorizationCode,
                0,
                appleIdCredential.AuthorizationCode.Length);

            // And now you have all the information to create/login a user in your system
        }
    },
    error =>
    {
        // Something went wrong
        var authorizationErrorCode = error.GetAuthorizationErrorCode();
    });
```

Quick login

This should be the **first thing to try** when the user first runs the application.

If the user has previously authorized the app to login with Apple, this will open a native dialog to re-confirm the login, and obtain an Apple User ID.

If the credentials were never given, or they were revoked, the Quick login will fail.



```
var quickLoginArgs = new AppleAuthQuickLoginArgs();

this.appleAuthManager.QuickLogin(
    quickLoginArgs,
    credential =>
    {
        // Received a valid credential!
        // Try casting to IAppleIDCredential or IPasswordCredential

        // Previous Apple sign in credential
        var appleIdCredential = credential as IAppleIDCredential;

        // Saved Keychain credential (read about Keychain Items)
        var passwordCredential = credential as IPasswordCredential;
    },
    error =>
    {
        // Quick login failed. Go to login screen
    });
```

Note that, if this succeeds, you will **ONLY** receive the Apple User ID (no email or name, even if it was previously requested).

IOS Keychain Support

When performing a quick login, if the SDK detects IOS Keychain credentials for your app, it will return those.

Just cast the credential to `IPasswordCredential` to get the login details for the user.

Checking credential status

This should be the first thing to check when the user starts the app, and there is an already logged user with an Apple user id.

Given an `userId` from a previous successful sign in. You can check the credential state of that user ID like so:

```
this.appleAuthManager.GetCredentialState(
    userId,
    state =>
    {
        switch (state)
        {
            case CredentialState.Authorized:
                // User ID is still valid. Login the user.
                break;

            case CredentialState.Revoked:
                // User ID was revoked. Go to login screen.
                break;

            case CredentialState.NotFound:
                // User ID was not found. Go to login screen.
                break;
        }
    },
    error =>
    {
        // Something went wrong
    });
```

Listening to credentials revoked notification

It may be that your user suddenly decides to revoke the authorization that was given previously. You should be able to listen to the incoming notification by registering a callback for it.

```
this.appleAuthManager.SetCredentialsRevokedCallback(result =>
{
    // Sign in with Apple Credentials were revoked.
    // Discard credentials/user id and go to login screen.
});
```

To clear the callback, and stop listening to notifications, simply set it to `null`

```
this.appleAuthManager.SetCredentialsRevokedCallback(null);
```

Nonce and State support for Authorization Requests

Both methods, `LoginWithAppleId` and `QuickLogin`, use a custom structure containing arguments for the authorization request.

An optional `Nonce` and an optional `State` can be set for both structures when constructing them:

```
// Your custom Nonce string
var yourCustomNonce = "RANDOM_NONCE_FORTHEAUTHORIZATIONREQUEST";
var yourCustomState = "RANDOM_STATE_FORTHEAUTHORIZATIONREQUEST";

// Arguments for a normal Sign In With Apple Request
var loginArgs = new AppleAuthLoginArgs(
    LoginOptions.IncludeEmail | LoginOptions.IncludeFullName,
    yourCustomNonce,
    yourCustomState);

// Arguments for a Quick Login
var quickLoginArgs = new AppleAuthQuickLoginArgs(yourCustomNonce, yourCustomState);
```

The `State` is returned later in the received Apple ID credential, allowing you to validate that the request was generated in your device.

The `Nonce` is embedded in the IdentityToken, included in the received Apple ID credential. It is important to generate a new random `Nonce` for every request. This is useful for services that provide a built in solution for **Sign In With Apple**, like Firebase

Some tentative guide is available for Firebase integration here

More info about State and Nonce can be found in this WWDC 2020 session (check at 2m35s)

FAQ

Does it support landscape orientations?

On **iOS 13.0**, Apple does not support landscape orientation for this feature. For more details, check this issue.

How can I Logout? Does the plugin provide any Logout option?

On **iOS 13** Apple does not provide any method to *"logout"* programatically. If you want to *"logout"* and re-test account creation, you need to revoke the credentials through settings.

Go to `Settings` => `Click your iTunes user` => `Password & Security` => `Apple ID logins`. There you can select the app and click on `Stop using Apple ID`.

After this, the credentials are effectively revoked, your app will receive a Credentials Revoked notification. This will allow you to re-test account creation.

I am not getting a full name, or an email, even though I am requesting them in the LoginWithAppleId call

This probably means that you already used Sign In with apple at some point. Apple will give you the email/name **ONLY ONCE**. Once the credential is created, **it's your app/game's responsibility to send that information somewhere**, so an account is created with the given user identifier.

If a credential was already created, you will only receive a user identifier, so it will work similarly to a Quick Login.

If you want to test new account scenarios, you need to revoke your app credentials for that Apple ID through the settings menu.

Is it possible to NOT request the user's email or full name?

Yes, just provide `LoginOptions.None` when calling `LoginWithAppleId` and the user will not be asked for their email or full name. This will skip that entire login step and make it more smooth. It is recommended if the user's email or full name is not used.

```
appleAuthManager.LoginWithAppleId(LoginOptions.None, credential => { ... }, error => { ... });
```

Does the plugin use UnitySendMessage?

No. The plugin uses callbacks in a static context with request identifiers using JSON strings. Callbacks are scheduled inside `AppleAuthManager`, and calling `Update` on it will execute those pending callbacks.

Why do I need to call Update manually on the AppleAuthManager instance?

Callbacks from iOS SDK are executed in their own thread (normally the main thread), and outside Unity's engine control. Meaning that you can't update the UI, or worse, if your callback throws an Exception (like a simple NRE), it will crash the Game completely.

It's recommended to update the instance of `AppleAuthManager` regularly in a MonoBehaviour of your choice.

What deserialization library does it use by default?

If you initialize the `AppleAuthManager` with the built-in `PayloadDeserializer`, it uses Unity JSON serialization system, so **no extra libraries are added**.

You can also implement your own deserialization by implementing an `IPayloadDeserializer`.

Any way to get a refresh token after the first user authorization?

It seems currently is not possible to do so. You can read more details here

I am getting a CFBundleIdentifier Collision error when uploading my app to the macOS App Store:

If you are experiencing an error like this when uploading your macOS app to the App Store

> The info.plist CFBundleIdentifier value 'com.lupidan.MacOSAppleAuthManager" of 'appname.app/Contents/Plugins/MacOSAppleAuthManager.bundle' is already in use by another application"

It probably means that your postprocess build script for macOS is not setup correctly.

You should call `AppleAuthMacosPostprocessorHelper.FixManagerBundleIdentifier` to fix the plugin's bundle identifier to a custom one for your app.

You can find more details about the bug here