Support for Apple Sign-in

At Worldwide Developers Conference (WWDC) in June, Apple announced a new product: Sign in with Apple. With the imminent release of iOS 13 on September 19, Apple has updated the App Store Review Guidelines and they now require any new applications that use third-party or social login services to offer Sign in with Apple as an equivalent option. Existing applications will be required to comply by April 2020. You can read more about this change on Apple's developer site here.

We know many Unity developers depend on third-party sign-in services. To make complying with these new guidelines easier, we have created a new asset store package. You can add the package to new or existing projects to leverage the new Sign in with Apple feature easily.

Below you will find the following:

1) A step by step guide of how to use the new asset store plugin
2) Links to important guidelines
3) For more advanced use cases, an overview of how you can perform server-side validation

Getting started

To get started with Sign in with Apple in we have created a new Asset Store package The purpose of this package is to make available the newly added iOS 13 API's required to use Sign in with Apple.

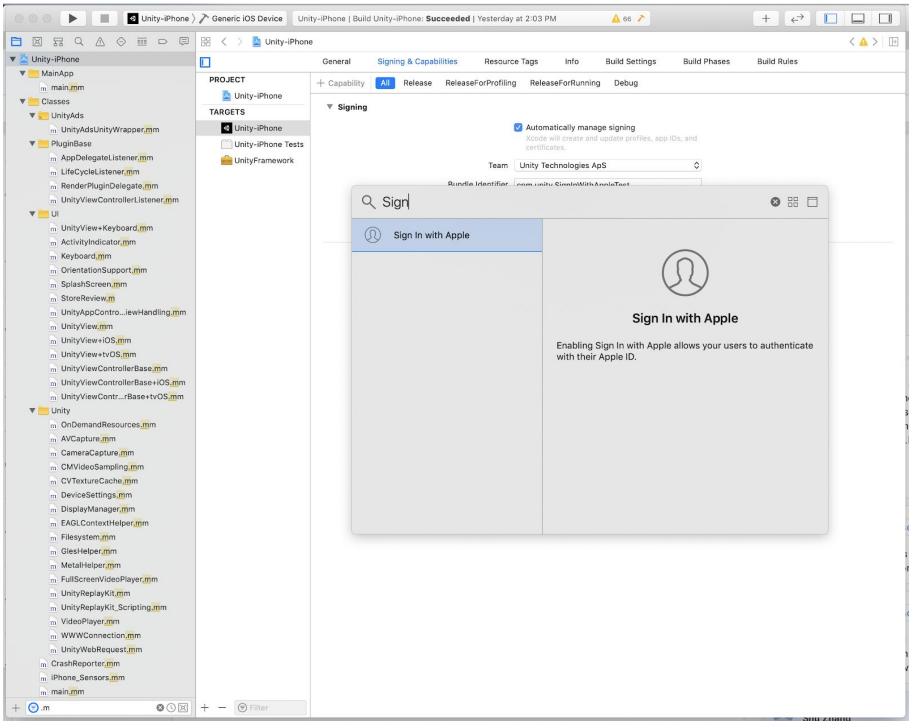
You will also need Xcode 11 which will work on either macOS 10.14 (Mojave) or 10.15 (Catalina) and a device that has iOS 13.0 installed. Downloads for Xcode and iOS 13 can be found here. It is also recommended that you read and review the Sign in With Apple Getting Started Guide. This guide covers Apple's Human Interface Guidelines and App Store Review Guidelines. Your application will also need to have the Sign in with Apple capability enabled in Apple's developer portal. Further instructions can be found here.

You can download the new package from the Unity Asset Store here.

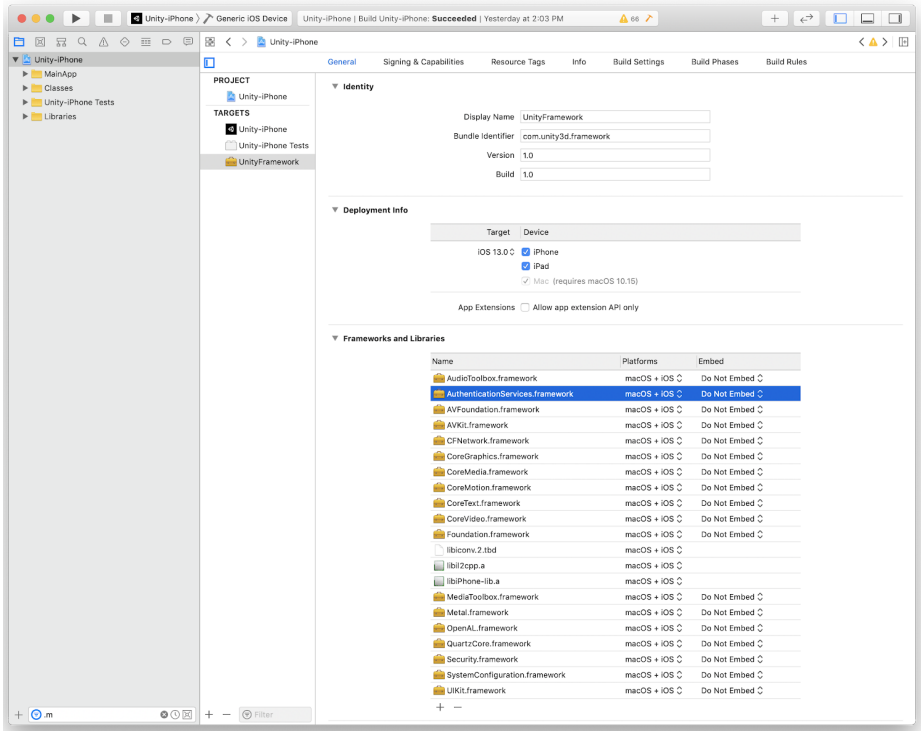
To use the package:

1) Import the package into an existing Unity project.

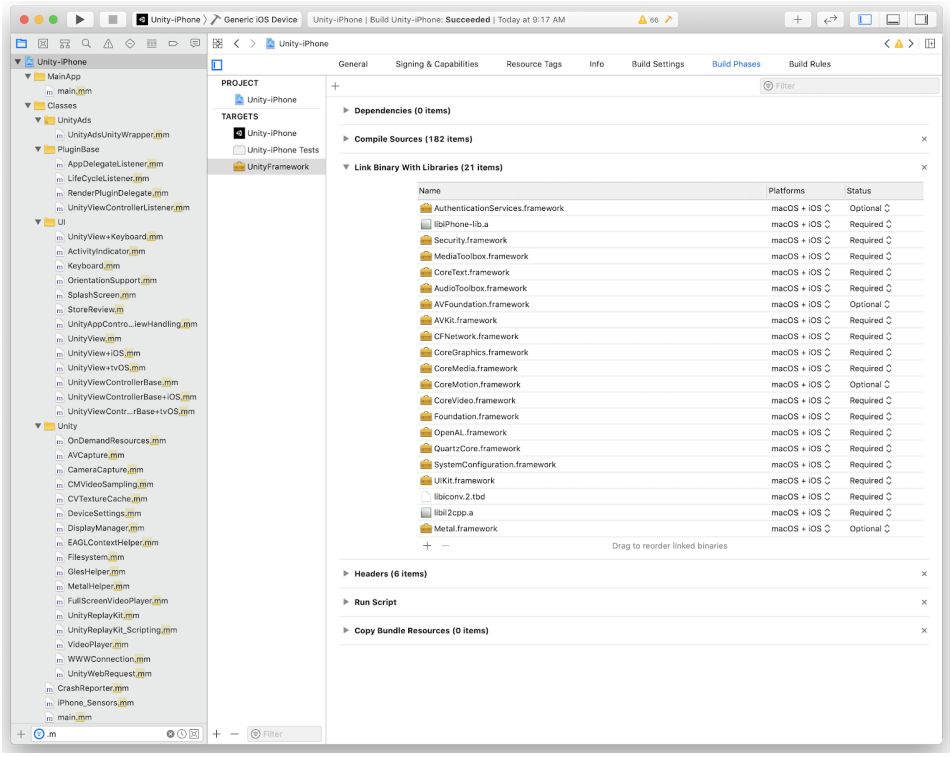
2) Create a script that has a callback to receive data about the login from Apple. See the sample script “SignInWithApple.cs” for an example of how the API is used.
3) Rebuild your Unity project and open the Xcode project created by Unity.
4) Entitlements and framework dependencies need to be configured within Xcode.
    a) In Target Settings the “Sign in with Apple” capability must be added. Click the button labeled “+ Capability” and choose “Sign in with Apple” as seen below.



    b) The AuthenticationServices framework must be added to the project. *You will need to mark this framework optional if you plan to target prior versions of iOS.* Select the UnityFramework target. Under the Frameworks and Libraries list is a + button. Click the button and choose the AuthenticationServices.framework.



c) To mark the framework as optional choose the Build Phases tab, expand the Link Binary with Libraries section. Find the AuthenticationServices framework and change the Status from “Required” to “Optional”.



Note: Xcode projects created with Unity 2019.2 or lower will not have the UnityFramework target. The AuthenticationServices framework should be added to the Unity-iPhone target instead.

5) With your project configured you can now extract any data made available in the callbacks to be used with the projects existing code base.

Server Side Validation

For games or applications which also require server side identity validation you can pass the identityToken to a server for validation.

Apple's identityToken is a JWT token which the client side cannot generate. To validate that JWT token is issued by Apple and intended to be used by your app, you must verify:

- Apply the standard JWT validation:
  - Validate that token is in valid format.
  - Validate that the token's signature is valid. The valid keys are found in the JWKS URL: https://appleid.apple.com/auth/keys
  - Validate that `iss` is `https://appleid.apple.com`
  - Validate that the token is not expired.
    - Make sure the `exp` claims must be after the current time.

- Check the audience of the JWT token is for your app.
    - The token you get back from Sign in with Apple uses your iOS app ID as the audience. Your server side needs to validate it to make sure it is intended to your app.

The Apple JWT payload claims are:

Claim	Column Name	Type	Example	Description
iss	Issuer	string	https://appleid.apple.com	The value is always https://appleid.apple.com
aud	Audience	string	com.unity.testApp	The audience of the token.
exp	Expiration Time	number	1568671600	The expiration time in epoch (seconds since 1970-01-01 00:00:00Z) of the token.
iat	Issued At	number	1568671000	The time in epoch at which the token is issued.
sub	Subject	string	001999.80b18c74c3264cad895d0eae181d8f50.1909	The user ID of the authenticated user.
c_hash	Code Hash	string	agyAh42GdE-O72Y4HUHypg	The hash of the authorization code. It's only used when you need to validate the authorization code.
email	Email	string	xxx@privaterelay.appleid.com	The email address of the user.
email_verified	Email Verified	string	true	Whether the email is verified. Note that it's a string JSON type.
auth_time	Auth Time	number	1568671000	The time in epoch at which the authentication happened.

There are JWT libraries available in most programming languages. These libraries can help you parse and validate the token.

If you use javascript, the following is an example of how to validate the token. It reads the token from stdin, and try to validate the token. Replace "your.app.id" with your real app ID from Apple.

```
const jwt = require('jsonwebtoken')
const jwksClient = require('jwks-rsa');
fs = require('fs');

var token = fs.readFileSync('/dev/stdin').toString().trim();
console.log(token);

var client = jwksClient({
  jwksUri: 'https://appleid.apple.com/auth/keys'
});

function getApplePublicKey(header, callback) {
  client.getSigningKey(header.kid, function (err, key) {
    var signingKey = key.publicKey || key.rsaPublicKey;
    callback(null, signingKey);
  });
}

jwt.verify(token, getApplePublicKey, null, function (err, decoded) {
  if (err) {
    console.error(err);
    process.exit(1);
  }
  if (decoded.iss !== "https://appleid.apple.com") {
    console.error("unexpected issuer (iss claim): ", decoded.iss);
    process.exit(1);
  }
  if (decoded.aud !== "your.app.id") {
    console.error("unexpected audience (aud claim): ", decoded.aud);
    process.exit(1);
  }
  console.log("Validated Apple token: ", decoded);
});
```