Table of Contents

Quick Start

Please note that the code provided in this page is *purely* for learning purposes and is far from perfect. Remember to null-check all responses!

Breaking Changes Notice

This package supports Google Sign In for Android 6.0+ (API level 23) and Sign in With Apple for iOS 14.0+. If you've just updated the package, it is recommended to check the *changelogs* for information on breaking changes.

Integrate Google Sign-In (Android)

Setup your app for Google Sign-In by following these steps (taken from google-signin-unity):

> ***Configuring the application on the API Console***
>
> To authenticate you need to create credentials on the API console for your application. The steps to do this are available on Google Sign-In for Android or as part of Firebase configuration. In order to access ID tokens or server auth codes, you also need to configure a web client ID.
>
> ***Get a Google Sign-In configuration file***
>
> This file contains the client-side information needed to use Google Sign-in. The details on how to do this are documented on the Developer website.
>
> Once you have the configuration file, open it in a text editor. In the middle of the file you should see the **oauth_client** section:

```
      "oauth_client": [
        {
          "client_id": "411000067631-
hmh4e210xxxxxxxxxx373t3icpju8ooi.apps.googleusercontent.com",
          "client_type": 3
        },
        {
          "client_id": "411000067631-
udra361txxxxxxxxxx561o9u9hc0java.apps.googleusercontent.com",
          "client_type": 1,
          "android_info": {
            "package_name": "com.your.package.name.",
            "certificate_hash": "7ada045ccccccccc677a38c91474628d6c55d03"
          }
```

```
        }
      ]
```

There are 3 values you need for configuring your Unity project:

1. The **Web client ID**. This is needed for generating a server auth code for your backend server, or for generating an ID token. This is the `client_id` value for the oauth client with client_type == 3.
2. The **package_name**. The client entry with client_type == 1 is the Android client. The package_name must be entered in the Unity player settings.
3. The keystore used to sign your application. This is configured in the publishing settings of the Android Player properties in the Unity editor. This must be the same keystore used to generate the SHA1 fingerprint when creating the application on the console.

   NOTE: The configutation file does not reference the keystore, you need to keep track of this yourself.

Scene Setup

Add an instance of `GoogleSignInManager` to your first scene (as it is a persistent singleton) and set `ServerClientId` to your Web client ID.

Sign In

To sign in at runtime, just call `SignIn()` and register to its callbacks (`OnSignedIn` and `OnSignInFailed`) or call `SignInAsync()` to get the results asynchronously:

```
using Uralstech.UMoth.GoogleSignIn;

public async void SignIn()
{
    (GoogleIdTokenCredential credential, GoogleSignInErrorCode failReason) = await
GoogleSignInManager.Instance.SignInAsync();
    if (credential == null)
    {
        Debug.LogError($"Failed to get credentials due to error: {failReason}");
        return;
    }

    Debug.Log($"Got credentials: {credential}");
}
```

Both `SignIn()` and `SignInAsync()` contain optional parameters to configure the operation. Please check the reference documentation for more info.

Sign Out

Like signing in, just call `SignOut()` and register to `OnSignedOut` and `OnSignOutFailed` or call `SignOutAsync()` to get the results asynchronously:

```
using Uralstech.UMoth.GoogleSignIn;

public async void SignOut()
{
    bool result = await GoogleSignInManager.Instance.SignOutAsync();
    if (!result)
        Debug.LogError("Could not sign out!");
    else
        Debug.Log("Signed out successfully.");
}
```

Firebase Integration

You can use the `IdToken` from signing in to create a Firebase Auth credential, like so:

```
using Uralstech.UMoth.GoogleSignIn;

(GoogleIdTokenCredential? result, GoogleSignInErrorCode failReason) = await
GoogleSignInManager.Instance.SignInAsync();
if (result is null)
{
    Debug.LogError($"Could not sign in due to error: {failReason}");
    return;
}

Credential fbCredential = GoogleAuthProvider.GetCredential(result.IdToken, null);

try
{
    AuthResult authResult = await
FirebaseAuth.DefaultInstance.SignInAndRetrieveDataWithCredentialAsync(fbCredential).Configur
eAwait(true);
    Debug.Log("User logged in successfully.");
}
catch (FirebaseException exception)
{
```

```
    Debug.LogException(exception);
}
```

Integrate Sign In with Apple (iOS)

This plugin doesn't require any additional setup for Sign In with Apple on iOS, but make sure you've set up everything on the Apple developer portal to enable the capability.

Scene Setup

Add an instance of `AppleIdSignInManager` to your first scene (as it is a persistent singleton).

Sign In

To sign in at runtime, just call `SignIn()` and register to its callbacks (`OnSignedIn` and `OnSignInFailed`) or call `SignInAsync()` to get the results asynchronously. You can also specify what user data you want from their AppleID:

```
using Uralstech.UMoth.AppleIdSignIn;

public async void SignIn()
{
    // If you only wanted the email, just pass in AppleIdScope.Email. You can also pass in AppleIdScope.None if you want neither.
    (AppleIdCredential credential, AppleIdSignInErrorCode failReason) = await AppleIdSignInManager.Instance.SignInAsync(AppleIdScope.FullName | AppleIdScope.Email);
    if (credential == null)
    {
        Debug.LogError($"Failed to get credentials due to error: {failReason}");
        return;
    }

    Debug.Log($"Got credentials: {credential}");
}
```

Both `SignIn()` and `SignInAsync()` contain optional parameters to configure the operation. Please check the reference documentation for more info.

Sign Out

Apple doesn't provide an API to programmatically sign the user out.

Check AppleID Credential State

You can check if the user has revoked authorization to their AppleID for the app using `GetCredentialState` and registering to its callback `OnGotCredentialState` or by calling `GetCredentialStateAsync` to get the results asynchronously. You will have to provide the user ID provided by Apple when you signed the user in. This example gets it from the current logged in Firebase Auth user:

```
using Uralstech.UMoth.AppleIdSignIn;

FirebaseUser user = FirebaseAuth.DefaultInstance.CurrentUser;
IUserInfo? appleIdData = user.ProviderData.FirstOrDefault(data => data.ProviderId
== "apple.com");

if (appleIdData != null)
{
    (AppleIdCredentialState state, string? errorDescription) = await
AppleIdSignInManager.Instance.GetCredentialStateAsync(appleIdData.UserId);
    Debug.Log($"Got credential state for current user, state: {state},
error: {errorDescription}");
}
```

Firebase Integration

You can use the `IdentityToken` and `AuthorizationCode` from signing in to create a Firebase credential. This example also creates a cryptographically secure nonce generated using `HashUtils`:

```
using Uralstech.UMoth;
using Uralstech.UMoth.AppleIdSignIn;

string rawNonce = HashUtils.RandomNonceString();
string nonceHash = HashUtils.Sha256Hash(rawNonce);

(AppleIdCredential? result, AppleIdSignInErrorCode failReason) = await
AppleIdSignInManager.Instance.SignInAsync(AppleIdScope.FullName |
AppleIdScope.Email, nonceHash);
if (result is null)
{
    Debug.LogError($"Could not sign in due to error: {failReason}");
    return;
}

// The nonce here is completely optional, use null if you don't want to use it.
Credential fbCredential = OAuthProvider.GetCredential("apple.com", result.IdentityToken,
rawNonce, result.AuthorizationCode);
```

```
try
{
    AuthResult authResult = await
FirebaseAuth.DefaultInstance.SignInAndRetrieveDataWithCredentialAsync(fbCredential).Configur
eAwait(true);
    Debug.Log("User logged in successfully.");
}
catch (FirebaseException exception)
{
    Debug.LogException(exception);
}
```