APPLE SIGN IN MANUAL

Contents

Introduction

This PDF manual is designed for you to use as a reference to the different Apple Sign In extension functions for iOS, tvOS and macOS, and as such does *not* contain a tutorial on how to set up the API in your games. We recommend that before doing *anything* with this extension, you take a moment to look over the official Apple Sign In API documentation, as it will familiarise you with many of the terms and concepts required to use the extension correctly:

- Apple Developer: Sign In With Apple

Note that this manual covers *both* the macOS and iOS/tvOS Sign In functions, as they **work exactly the same way and have only been separated into two extensions so that you can use one or the other or both as required**. This means that you may need to do certain checks using the `os_type` variable to call the correct function for the current platform the game is running on, but the bulk of the code will be the same regardless. The examples in this manual are based on the iOS/tvOS functions and constants, so for macOS you would simply swap (or duplicate, if developing for both platforms) the function/constant names for the macOS versions and the Sign In functionality should work exactly the same.

The Asynchronous System Event

When using the Apple Sign In extension in your projects, you will be calling different functions that will trigger "callbacks" from the Apple API. What this means is that certain functions will be run but won't return a result until sometime in the future - which could be the next step, or it could be a few seconds later.

This result, when it comes, is called the "callback" and is Apple's Sign In API responding to something you've done. This callback is dealt with in the **Asynchronous System Event**.

This event will always have a DS map in the GML variable **async_load**, and this map can be parsed to get the required information. Each function will generate different callbacks, but they will all have the following key in common:

- **"id"** – This is the event ID key and it will hold a CONSTANT with the ID of the event that has been triggered. For example, if it's an event for authorising a user, then the constant will be `applesignin_signin_response / mac_applesignin_signin_response`. See the different functions for details about the constants returned for each.

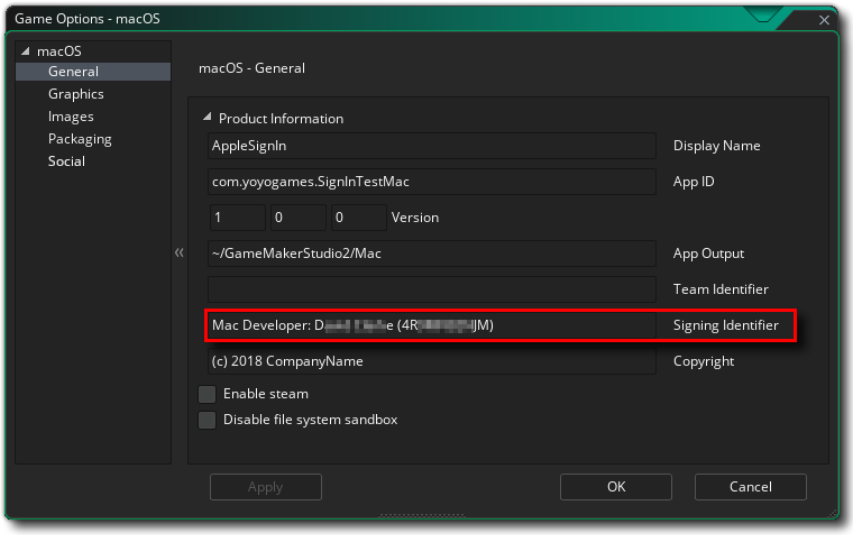
The rest of the key/value pairs in the map will depend on the function that triggered the Async Event and the ID of the event, and you should check the individual functions listed in the rest of this manual for exact details.

Extension Functions

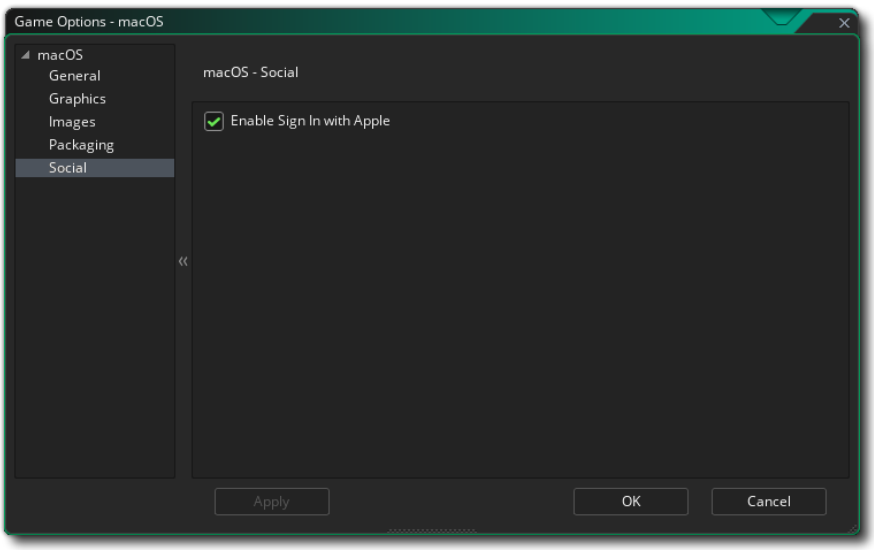
The rest of this manual contains a reference guide to all the functions used by the Apple Sign In Extension, along with any constants that they may use or return and examples of code that use them. Some of the examples are **Extended Examples** that also show code from callbacks in the Asynchronous System Event.

Use of this extension is required by Apple whenever your app provides a third-party sign in option (for example, Facebook), and is available for iOS/tvOS 13 and above and macOS 10.15 (Beta) and above. To test your apps, you'll need to use the latest beta version of Xcode 11 and update your devices to the latest OS versions (which may require you to update to beta versions).

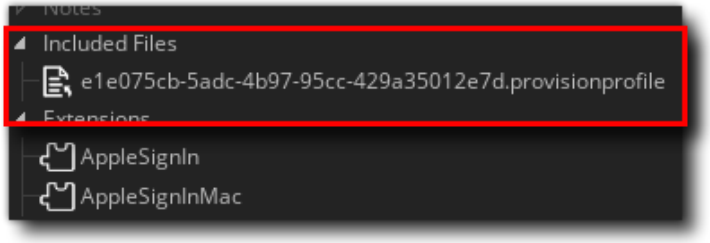
Additionally, for **macOS** to use this extension you will be required to provide a **Signing Identifier** in the **macOS Game Options**:



And you will also need to check the "Enable Sign In With Apple" checkbox in the **Social** section:



Also note that if you are compiling for macOS using the **VM** and *not* the YYC, then Xcode will not be used and so you need to take an extra step to ensure that your game will function correctly. This requires you to retrieve a **Provisioning Profile** for your game (see here for more information) and add it into GameMaker Studio 2 as an **included file**:



This profile should be valid for the App ID, Team Identifier and Signing Identifier used in the Game Options.

Once you have everything setup, it's simply a case of adding a button object into your game and having it call the appropriate functions when pressed, and then parsing the Asynchronous System Event to get the necessary data from the callback.

> **IMPORTANT!** *This functionality* ***cannot be tested by running the game from the GameMaker Studio IDE*** *and it is required that you create an executable for the target platform to properly test.*

AppleSignIn_Init / Mac_AppleSignIn_Init

Description

This function will initialise the Apple Sign In API and **is called automatically by the extension**. As such, you should *not* be using it ever in your game code as it is not required.

Syntax

```
AppleSignIn_Init();
Mac_AppleSignIn_Init();
```

Returns

N/A

Example

N/A

Mac_AppleSignIn_Final

Description

This function will finalise the Apple Sign In API on macOS and **is called automatically by the extension**. As such, you should *not* be using it ever in your game code as it is not required.

Syntax

```
Mac_AppleSignIn_Final();
```

Returns

N/A

Example

N/A

AppleSignIn_AddScope / Mac_AppleSignIn_AddScope

Description

This function can be used to add a scope for the sign in request to request additional pieces of user data. The additional data you can request are for an email address and the user's full name, and these are requested using the following constants:

Constant	Actual Value	Description
applesignin_scope_fullname mac_applesignin_scope_fullname	"fullname"	Requests the users full name.
applesignin_scope_email mac_applesignin_scope_email	"email"	Requests the users email address.

Note that this function **must be called before authorisation**, so you should call it before any calls to the authorisation functions AppleSignIn_AuthoriseUser() / Mac_AppleSignIn_AuthoriseUser(). The callback for those functions will contain additional data depending on the scopes added.

Syntax

```
AppleSignIn_AddScope(scope);
Mac_AppleSignIn_AddScope(scope);
```

Argument	Description	Data Type
Scope	One of the scope constants listed above	String

Returns

N/A

Example

The following code example would go in the Step event of a button object to check for a user wanting to sign in using the Apple Sign In API:

```
if mouse_check_button_pressed(mb_left)
    {
    if instance_position(mouse_x, mouse_y, id)
        {
        AppleSignIn_AddScope(applesignin_scope_fullname);
        AppleSignIn_AddScope(applesignin_scope_email);
        AppleSignIn_AuthoriseUser();
        }
    }
```

AppleSignIn_ClearScopes / Mac_AppleSignIn_ClearScopes

Description

This function is used to clear the scope(s) set for the user when requesting Sign In authorisation. In general this is not required at any time, but may be useful if you wish to – for example – have a sign in with email and full name permissions. The user may cancel this as they don't want to share that data, so you can clear the permissions using this function and then let them sign in again with only basic permission scopes.

Syntax

```
AppleSignIn_ClearScopes();
Mac_AppleSignIn_ClearScopes();
```

Returns

N/A

Example

The code below shows an example of the Asynchronous Social Event after the user has pressed a button to sign in to your app and the function `AppleSignIn_AuthoriseUser()` has been called:

```
var _event_id = async_load[? "id"];
switch (_event_id)
    {
    case applesignin_signin_response:
        global.appleSignInState = -1;
        var _json = async_load[? "response_json"];
        if _json != ""
            {
            var _map = json_decode(_json);
            if _map[? "success"] == true
                {
                show_debug_message("Apple Sign In Succeeded");
                global.appleSignInState = applesignin_state_authorized;
                }
            ds_map_destroy(_map);
            }
        if global.appleSignInState == -1
            {
            AppleSignIn_ClearScopes();
            }
        break;
    }
```

AppleSignIn_AuthoriseUser / Mac_AppleSignIn_AuthoriseUser

Description

This function will authorise a user using the Apple Sign In API and their Apple ID credentials. The function will trigger an **Asynchronous System Event** where the `async_load` DS map "**id**" key will be (depending on the platform) the constant **`applesignin_signin_response`** / **`mac_applesignin_signin_response`**. The `async_load` map will also contain the key "**response_json**", which can then be decoded into another DS map using the function `json_decode()`. This map will have the following keys:

- "**success**" – This will be `true` or `false` depending on whether authorisation was granted or not. If this is `false`, none of the other keys listed will be present.
- "**identityToken**" – This will hold the unique identity token string for the session.
- "**userIdentifier**" – This will hold the unique user identifier string.
- "**realUserStatus**" – This will hold one of the following constants:

Constant	Actual Value	Description
applesignin_realuserstatus_likelyreal mac_applesignin_realuserstatus_likelyreal	5002 5102	The user is likely a real person.
applesignin_realuserstatus_unknown mac_applesignin_realuserstatus_unknown	5001 5101	The system hasn't determined whether the user might be a real person.
applesignin_realuserstatus_unsupported mac_applesignin_realuserstatus_unsupported	5000 5100	This indicates that the IAP product ID has already been added to the internal product list.

- "**authCode**" – This will hold the unique authorisation code string.

Cont…/

Additionally, if you have used the AppleSignIn_AddScope() / Mac_AppleSignIn_AddScope() function before calling the authorise function, then you may have the following additional keys – first, for the applesignin_scope_email / mac_applesignin_scope_email scope:

- "**email**" – This will be the users email address (as a string).

Then for the applesignin_scope_fullname / mac_applesignin_scope_fullname scope (note that these are all strings if the data is available, and empty strings "" if not):

- "**namePrefix**" – The user's name prefix
- "**phoneticRepresentation**" – A phonetic representation of the name
- "**givenName**" – The first name of the user
- "**middleName**" – The middle name of the user
- "**nameSuffix**" – Any suffix that may be applicable to the name
- "**nickname**" – The user's nickname
- "**familyName**" – The user's family (second) name

Syntax

```
AppleSignIn_AuthoriseUser();
Mac_AppleSignIn_AuthoriseUser();
```

Returns

N/A

Cont…/

Example

The authorise function would normally be called from a button or a controller object in your game, and would generate an Asynchronous Social Event that can be dealt with in the following way:

```
var _event_id = async_load[? "id"];
switch (_event_id)
    {
    case applesignin_signin_response:
        global.appleSignInState = -1;
        var _json = async_load[? "response_json"];
        if _json != ""
            {
            var _map = json_decode(_json);
            if _map[? “success”] == true
                {
                show_debug_message("Apple Sign In Succeeded");
                global.appleSignInState = applesignin_state_authorized;
                }
            ds_map_destroy(_map);
            }
        break;
    }
```

Note that we have a global variable to store the sign in state for future reference, where -1 means not signed in / no action has been taken, and then we can use one of the following extension constants to check for other states as required:

Constant	Actual Value	Description
applesignin_state_authorized mac_applesignin_state_authorized	100 1100	The user has been authorized and has a valid session
applesignin_state_revoked mac_applesignin_state_revoked	101 1101	The user's credentials have been revoked and the session terminated
applesignin_state_not_found mac_applesignin_state_not_found	102 1102	The user does not appear to have a session with your app

AppleSignIn_GetCredentialState / Mac_AppleSignIn_GetCredentialState

Description

With this function you can retrieve the credential sign in state for your game. You supply the identity token string for the session (which is returned as part of the callback when you use the function AppleSignIn_AuthoriseUser / Mac_AppleSignIn_AuthoriseUser()) and the function will trigger an Asynchronous System Event where the async_load DS map "**id**" key will be **applesignin_credential_response**. The async_load map will then have a further key "**response_json**", which will hold a JSON string which can be parsed into a DS map using the function json_decode(). This map will have the key "**status**" which will be one of the following constants:

Constant	Actual Value	Description
applesignin_state_authorized mac_applesignin_state_authorized	100 1100	The user has been authorized and has a valid session
applesignin_state_revoked mac_applesignin_state_revoked	101 1101	The user's credentials have been revoked and the session terminated
applesignin_state_not_found mac_applesignin_state_not_found	102 1102	The user does not appear to have a session with your app

Syntax

```
AppleSignIn_GetCredentialState(token);
Mac_AppleSignIn_GetCredentialState(token);
```

Argument	Description	Data Type
token	The session identity token.	String

Returns

N/A

Cont…/

Example

The following code shows an example of how the callback response for this function would be parsed in the Asynchronous Social Event:

```
var _event_id = async_load[? "id"];
switch (_event_id)
    {
    case applesignin_credential_response:
        global.appleSignInState = -1;
        var _json = async_load[? "response_json"];
        if _json != ""
            {
            var _map = json_decode(_json);
            global.appleSignInState = _map[? “state”];
            ds_map_destroy(_map);
            }
        break;
    }
```

Mac_AppleSignIn_RegisterWindow

Description

This **macOS only** function must be used before calling any other function to register the game window for authorisation UI overlays.

Syntax

```
Mac_AppleSignIn_RegisterWindow();
```

Returns

N/A

Example

The following code example would go in the Step event of a button object to check for a user wanting to sign in using the Apple Sign In API:

```
if mouse_check_button_pressed(mb_left)
    {
    if instance_position(mouse_x, mouse_y, id)
        {
        Mac_AppleSignIn_RegisterWindow();
        Mac_AppleSignIn_AddScope(applesignin_scope_fullname);
        Mac_AppleSignIn_AddScope(applesignin_scope_email);
        Mac_AppleSignIn_AuthoriseUser();
        }
    }
```

RegisterCallbacks

Description

This is an internal function for **macOS only**. This function should never be called in your code, and you should not edit or change anything about it otherwise the extension may no longer work.

Syntax

N/A

Returns

N/A

Example

N/A