# Apple GameCenter Extension

## Contents

## Introduction

Game Center lets players build an identity across Apple platforms and enables features like the Game Center leaderboards, achievements, multiplayer functionality, dashboard, and more. Add features within your custom user interface or take advantage of the updated Game Center user interface (website)

Leaderboards let players participate in new competitions within your game and challenge other players to beat their score. Players see how they rank among global players, as well as their Game Center friends. You can even create a friends-only leaderboard in your custom UI. Classic leaderboards are ongoing and maintain a player's score forever. Use these to showcase scoring for overall or lifetime events, such as all-time number of matches won or the shortest time taken to complete a level. Recurring leaderboards are short lived and expire, then repeat after a set time, such as daily, monthly, or weekly. Use recurring leaderboards for timed live events — for example, to show the most matches won in a week. You can use one or both leaderboard types in your game.

Achievements are special milestones that indicate when a player has successfully reached a particular goal in your game. You decide which achievements are relevant to your game and worth offering to your players. Players see achievements as locked, in progress, or completed. You can also create hidden achievements, which only appear once completed. Consider using hidden achievements to prevent spoilers in your game and to surprise and delight players. You can provide up to 100 achievements, each of which can award up to 100 points, and your game can award up to 1,000 points in total. Keep these limits in mind when releasing the initial version of your game, as you may want to add new achievements in future updates. For variety, consider creating a set of achievements that require dedication and a range of skills to complete.

## Extension's Features

- Sign in to Apple GameCenter
- Submit to and show specific leaderboards
- Show multiple leaderboards
- Report and show achievements
- Create save slots
- Save/Delete save slots
- Query all save slots

## Setup

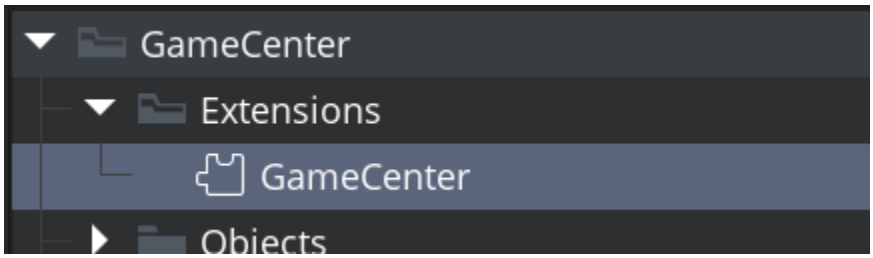
The Apple GameCenter extension is to be used alongside your Apple Developer account (web page). All the required personal leaderboard ids and achievement ids should be managed from there.

1. For creating leaderboards follow this guide: Adding Leaderboards

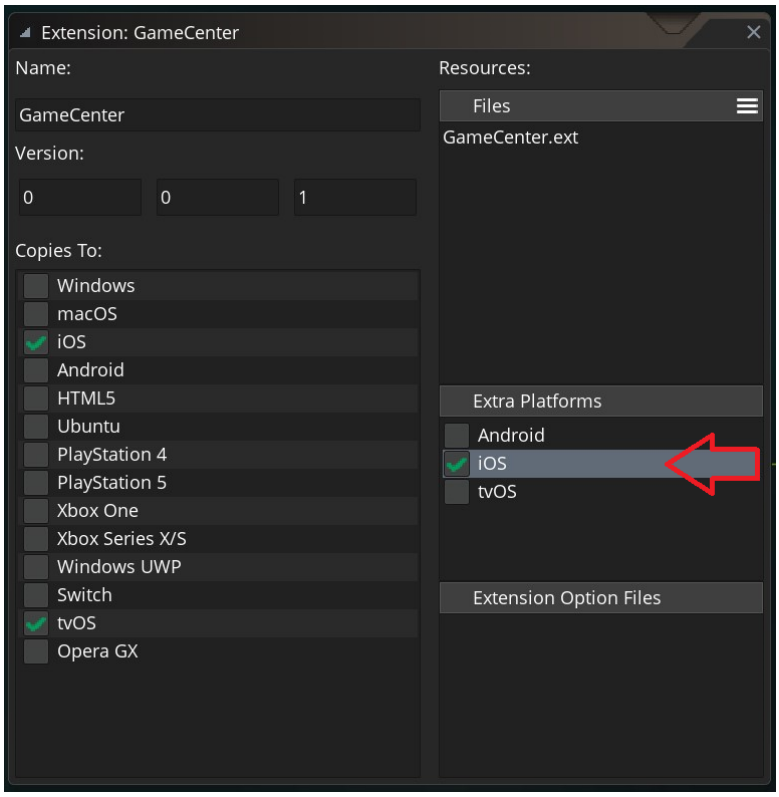
2. For creating achievements follow the guide: Adding Achievements

3. If you intend to use Save Games functionality you need to create a new iCloud container using your apple developer account dashboard (dashboard link).

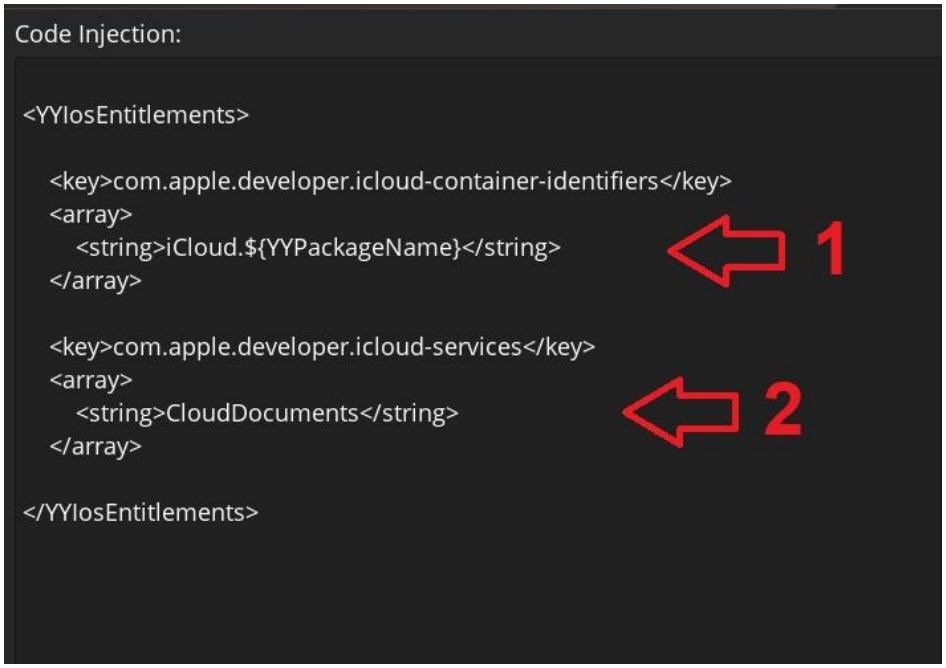
   This also needs to be set up inside the project itself, following these steps:




   double click the **GameCenter** extension icon on your project;




   click **iOS** line under **Extra Platforms** this will bring you to the code injection panel;




in this panel you need to add/replace **(1)** which will configure the XCode project to use the name of the iCloud container you previously created and **(2)** this will make sure the iCloud Drive services permissions are automatically added.

# Quick Start Guide

This section aims to deliver an easy and simple set of examples that should help you migrate your project from a previous version of the GameCenter extension to its latest version. The first thing to notice is that GameCenter functionality used to be part of the runtime and all the included functions were named "**achievement_***"; these were renamed to "**GameCenter_***" and also separated into categories to better organize them and provide a cleaner look to the code.

## Signing In/Out

The first thing to look for when creating an Apple GameCenter GMS2 project is signing in to a GameCenter account.

### [Function Mappings]

Even though not all old functions map perfectly into the new API, the following list should get you started with some of the changes (check the section below for more details):

- **GameCenter_LocalPlayer_IsAuthenticated()** ⇔ achievement_login_status()
- **GameCenter_LocalPlayer_Authenticate()** ⇔ achievement_login()
    - triggers the event of type "GameCenter_Authenticate" (more details)
- **GameCenter_LocalPlayer_GetInfo()** gets information on the local player.

### [Old Version]

```
// In the previous version you would check the achievement status and if not
// signed in, you were required to log in.
if (!achievement_login_status()) {
      achievement_login();
}
```

This function would then trigger a callback ASYNC SOCIAL EVENT, where the async_load map would contain an "id" key of **achievement_our_info** and also some account information under "name" and "playerId".

### [New Version]

```
// In the new version you can use the following code to check connection and
// connect to the GameCenter account.
if (!GameCenter_LocalPlayer_IsAuthenticated()) {
      GameCenter_LocalPlayer_Authenticate();
}
```

This function will then trigger a callback ASYNC SOCIAL EVENT, where the async_load map would contain a “type” key with the value “GameCenter_Authenticate” and a “success” key that can have a value of 1 or 0 (depending on whether or not the function was successful).

```
// After signing in if you need to query player information, you can do so
// by using the method 'GameCenter_LocalPlayer_GetInfo'.
var playerInfoJSON = GameCenter_LocalPlayer_GetInfo();
var playerInfo = json_parse(playerInfoJSON);
```

The player information after being parsed into a struct contains a lot of details about the account (see PlayerJSON).

## Leaderboards

One of the features provided by Apple GameCenter API is creation and the interaction with leaderboards. Leaderboards can be created using the Apple Developer Dashboard (see Setup) and you can see and submit values to them during runtime.

### [Function Mappings]

Even though not all old functions map perfectly into the new API, the following list should get you started with some of the changes (check the section below for more details):

- **GameCenter_Leaderboard_Submit(...)** ⇔ achievement_post_score()
    - triggers the event of type "GameCenter_Leaderboard_Submit" (more details)
- **GameCenter_PresentView_Leaderboards()** ⇔ achievement_show_leaderboards()
- **GameCenter_PresentView_Leaderboard(...)** shows a specific leaderboard.

### [Old Version]

```
// In the previous version you would use the leaderboards as follows.
// To submit a value to a leaderboard you would do:
achievement_post_score(global.leaderboardId, score);
```

This function would then trigger a callback ASYNC SOCIAL EVENT, where the async_load map would contain an "id" key of **achievement_post_score_result** with some other keys with information regarding your current post to the leaderboard.

```
// Then if you wanted to see the leaderboards you could just use the following
// line of code.
achievement_show_leaderboards();
```

### [New Version]

```
// In the new version you can submit to a leaderboard using the code sample:
GameCenter_Leaderboard_Submit(global.leaderboardId, score);
```

This function works asynchronously and triggers an Async Social Event callback upon completion with submission result information (GameCenter_Leaderboard_Submit).

```
// If you need to show the leaderboard to the user you can now do so with the
// help of one of two functions:

// Show a single leaderboard (GameCenter_PresentView_Leaderboard)
GameCenter_PresentView_Leaderboard(leaderboardId, timeScope, playerScore);

// Show all the leaderboards
GameCenter_PresentView_Leaderboards();
```

## Achievements

One of the features provided by Apple GameCenter API is creation and the interaction with achievements. Achievements can be created using the Apple Developer Dashboard (see Setup) and you can see and submit values to them during runtime.

**[Function Mappings]**

Even though not all old functions map perfectly into the new API, the following list should get you started with some of the changes (check the section below for more details):

- **GameCenter_Achievement_Report(...)** ⇔ achievement_post(...)
  - triggers the event of type "GameCenter_Achievement_Report" (more details)
- **GameCenter_Achievement_ResetAll()** ⇔ achievement_resetl()
  - triggers the event of type "GameCenter_Achievement_ResetAll" (more details)
- **GameCenter_PresentView_Achievements()** ⇔ achievement_show_achievements()
- **GameCenter_PresentView_Achievement(...)** shows a specific leaderboard.

**[Old Version]**

In previous versions you could reveal, increment, post and show in-game achievements using the functions given below:

```
// After successfully connecting to the Google Play account you could perform
// the following operations:
achievement_reveal(achievementId);
achievement_post(achievementId, percent);

achievement_increment(incrementalAchievementId, amount);
achievement_show_achievements();
```

The code above would first reveal the achievement to the user (if it was a hidden achievement), then you could post a value to an achievement which would be a percentage value (0 - incomplete, 100 - complete); you could also increment the achievement value (if it was an incremental achievement) and finally you could show the achievements to the user.

**[New Version]**

In the new version you can report achievement completion using the following code:

```
GameCenter_Achievement_Report(achievementId, percent);
GameCenter_PresentView_Achievement(achievementId); // Show single
GameCenter_PresentView_Achievements(); // Show all
```

The migration in the example above is very simple and straightforward. Please take into consideration that the code above will produce Async Social event callbacks with additional information regarding the task completion (more details).

## Save Games **[NEW]**

This functionality is completely new and as such there is no migration guide, however you can experiment with it using the provided demo sample. This section aims to deliver a clean explanation of the code layout used for the demo.

**NOTE**: This feature will only work if the end user's iOS device iCloud Drive is turned on. This feature also requires the developer's project to have a dedicated container defined for the app (defined on the Apple's CloudKit Dashboard).

**[Objects]**

1. Each component used in the sample is represented by an object (stored inside the folder: **GameCenter → Objects → SavedGames**)
2. The object **Obj_GameCenter_SavedGames** is where the ASYNC SOCIAL EVENT callbacks are handled.
3. The object **Obj_GameCenter_Point** is used as a placeholder representation of game data to be saved.
4. The object **Obj_GameCenter_Slot** represents a saved slot that can be clicked to enter slot edit mode.
5. The remaining objects inside the folder are buttons that the user can interact with.

**[Demo Workflow]**

1. When you create a new slot (**New Save Slot** button) a prompt dialog to name the slot is displayed. The callback to handle the returned value and entering slot edit mode is inside the **Obj_GameCenter_SavedGames** dialog async event.
2. In the edit mode we can move objects around and tap the icon to change it. After editing we can close edit mode ignoring new changes to the data or **save** the changes (GameCenter_SavedGames_Save), optionally we can also **delete** the slot (GameCenter_SavedGames_Delete).
3. After saving the new slot we **fetch** (GameCenter_SavedGames_Fetch) the slot data from Apple GameCenter servers.
4. The newly created slot will now appear on screen. If you click it, we will request to **get** its data (GameCenter_SavedGames_GetData) and enter its edit mode.

# Functions

Some of the provided functions generate **Social Async** callbacks. In these cases the extension populates the **async_load** map with a "**type**" key that will contain the specific identifier of a given callback. All GameCenter related callbacks start with "*GameCenter_*".

## LocalPlayer Functions

### `GameCenter_LocalPlayer_Authenticate()`

Description: This function requests the API for user authentication with GameCenter credentials. When using the Apple GameCenter extension you need to call this function before using any other functions. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback upon completion.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_Authenticate"

success: {*boolean*} whether or not the task succeeded.

### `GameCenter_LocalPlayer_IsAuthenticated()`

Description: This function checks if the current logged in user is authenticated or not.

Returns: {*boolean*} whether or not authentication succeeded.

### `GameCenter_LocalPlayer_IsUnderage()`

Description: This function checks if the current logged in user is underage or not.

Returns: {*boolean*} whether or not the current player is underaged.

### `GameCenter_LocalPlayer_GetInfo()`

Description: This function queries the Apple GameCenter server for information on the current player. The function will return a json formatted string that can be parsed into a struct using **json_parse**.

Returns: {PlayerJSON} A **json formatted string** containing player information.

## Achievements Functions

`GameCenter_Achievement_Report(achievementId, percent)`

Description: This function requests the Apple GameCenter API to update the completion percentage of the given achievement. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback upon completion.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

> type: "GameCenter_Achievements_Report"
>
> success: {*boolean*} whether or not the task succeeded.
>
> achievement_id: {*string*} The unique name of the achievement.

`GameCenter_Achievement_ResetAll()`

Description: This function resets all previously earned/reported achievements. This function is for debug purposes only and should not be used in production, unless specifically required..

Triggers: ASYNC SOCIAL EVENT

> type: "GameCenter_Achievements_ResetAll"
>
> success: {*boolean*} whether or not the task succeeded.

`GameCenter_Achievement_Load()`

Description: This function will load data from all the achievements the player has progressed on.

Triggers: ASYNC SOCIAL EVENT

> type: "GameCenter_Achievements_Load"
>
> success: {*boolean*} whether or not the task succeeded.
>
> data: {Array.AchievementJSON} A **json formatted string** containing an array of AchievementJSON information (can be parsed into a struct using json_parse)

## Leaderboard Functions

`GameCenter_Leaderboard_Submit(leaderId, score)`

Description: This function requests the Apple GameCenter API to submit a score to a given leaderboard. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when the task is resolved.

Params:

*{string}* **leaderId**: the unique identifier of the leaderboard.

*{real}* **score**: the value to be submitted to the leaderboard (remember that only the highest score value is displayed in the leaderboard).

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_Leaderboard_Submit"

success: {*boolean*} whether or not the task succeeded.

## Save Game Functions

### SavedGames Async Callbacks

Some GameCenter functionalities can occur from more than one iOS device with the same account; this can lead to conflicts while handling saved games. With that in mind the Apple GameCenter extension provides two additional callbacks that can occur at any point in time.

Description: This event will be triggered if a conflict was found while writing saved games (conflicts can occur if you have the same game open in different devices at the same time)

ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_HasConflict"

conflict_ind: {*real*} the unique identification index of the current conflict.

slots: {*string*} **json formatted** string with an **array** of SlotJSON.

Description: This event will trigger when the saved game gets modified you can use it to check if the game was saved from another device. If this event triggers and you didn't perform a GameCenter_SavedGames_Save call then this means the game was saved from elsewhere.

ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_DidModify"

slot: {*string*} **json formatted** string following the SlotJSON structure.

player: {*string*} **json formatted** string following the PlayerJSON structure.

`GameCenter_SavedGames_Fetch()`

Description: This function requests the Apple GameCenter API to fetch all the existing saved slots. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when the task is resolved.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_Fetch"

success: {*boolean*} whether or not the task succeeded.

slots: {string} **json formatted** string with an **array** of SlotJSON.

`GameCenter_SavedGames_Save(slotName, data)`

Description: This function requests the Apple GameCenter API to save a data string to a save slot, if the slot doesn't exist it gets created (if it exists data is overwritten). The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when the task is resolved.

Params:

*{string}* **slotName**: the unique identifier of the save game slot.

*{string}* **data**: the data string to be saved. Note that you can save complex data by using structs/arrays/maps/lists and converting them to strings.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_Save"

success: {*boolean*} whether or not the task succeeded.

name: the unique identifier of the save game slot.

slot: {string} **json formatted** string following the SlotJSON structure.

`GameCenter_SavedGames_Delete(slotName)`

Description: This function requests the Apple GameCenter API to delete an existing save slot with the given name. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when the task is resolved.

Params:

*{string}* **name**: the unique identifier of the save game slot.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_Delete"

success: {*boolean*} whether or not the task succeeded.

`GameCenter_SavedGames_GetData(slotName)`

Description: This function requests the Apple GameCenter API to get the data string saved inside a given save slot. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when the task is resolved.

Params:

*{string}* **slotName**: the unique identifier of the save game slot.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_GetData"

success: {*boolean*} whether or not the task succeeded.

**<if successful>**

data: {*string*} the data stored inside the save slot.

`GameCenter_SavedGames_ResolveConflict(conflictId, data)`

Description: This function requests the Apple GameCenter API to perform a conflict resolution given a conflict id. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when the task is resolved.

Params:

*{real}* **conflictId**: the unique identifier of the conflict.

*{string}* **data**: a string containing the data you want to save (note you can use json formatted string to store data).

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_SavedGames_ResolveConflict"

success: {*boolean*} whether or not the task succeeded.

conflict_ind: {*real*} the unique identification index of the resolved conflict.

## PresentView Functions

`GameCenter_PresentView_Default()`
Description: This function displays the general Apple GameCenter overlay screen with information on achievements and leaderboards. This overlay will trigger an ASYNC SOCIAL EVENT when closed.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_PresentView_DidFinish"

`GameCenter_PresentView_Achievement(achievementId)`
Description: This function displays the Apple GameCenter achievement overlay screen with the information of the specified achievement. This overlay will trigger an ASYNC SOCIAL EVENT when closed.

Params:

*{string}* **achievementId**: The unique achievement identification string.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_PresentView_DidFinish"

`GameCenter_PresentView_Achievements()`
Description: This function displays the Apple GameCenter achievement overlay screen with all your achievement information. This overlay will trigger an ASYNC SOCIAL EVENT when closed.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_PresentView_DidFinish"

`GameCenter_PresentView_Leaderboard(leaderId, timeScope, playerScope)`
Description: This function displays the Apple GameCenter leaderboard overlay screen with the information of a specific leaderboard. This overlay will trigger an ASYNC SOCIAL EVENT when closed.

Params:

*{string}* **leaderId**: The unique leaderboard identification string.

*{Time Scope}* **timeScope**: The time scope to be displayed.

*{Player Scope}* **playerScope**: The player scope to be displayed.

Returns: N/A

`GameCenter_PresentView_Leaderboards()`

Description: This function displays the Apple GameCenter achievement overlay screen with all your leaderboards information. This overlay will trigger an ASYNC SOCIAL EVENT when closed.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GameCenter_PresentView_DidFinish"

# JSON Structs

## PlayerJSON

→ alias: A string the player chooses to identify themself to other players.
→ displayName: A string to display for the player.
→ playerID: A unique identifier for a player of the game.

## AchievementJSON

→ identifier: A string that identifies the current achievement.
→ percentComplete: The complete percentage of the achievement.
→ isComplete: Whether or not the achievement was completed.
→ showsCompletionBanner: Whether or not the achievement shows a completion banner.
→ player: a PlayerJSON struct.
→ lastReportedDate: A GameMaker compliant date from when the last report occurred.

## SlotJSON

→ playerID: A unique identifier for a player of the game.
→ deviceName: The name of the device that the player used to save the game.
→ modificationDate: The date when you saved the game data or modified it.
→ name: The name of the saved game.

# Constants

## Time Scope

GameCenter_Leaderboard_TypeScope_Today: 0
GameCenter_Leaderboard_TypeScope_Week: 1
GameCenter_Leaderboard_TypeScope_AllTime: 2

## Player Scope

GameCenter_Leaderboard_PlayerScope_Global: 0
GameCenter_Leaderboard_PlayerScope_FriendsOnly: 1