Google Play Services Extension

Contents

Extension’s Features

- Sign in and out or Google Play Account
- Revoke login permissions
- Submit and show specific leaderboards
- Show multiple leaderboards
- Handle achievements:
    - Unlock (one-time achievements)
    - Incremental (progress achievements)
    - Reveal (hidden achievements)
- Open save slots
- Submit/Delete save slots
- Query all save slots
- Show GooglePlayUI for saved slots

Setup

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

> **[IMPORTANT]** before using this extension make sure to correctly set up your Android application following the steps on this article. It's also important to note that this is a reference demo meaning it cannot be played as is, as it would need our **.keystore** and our **Google Services ID** which cannot be included with the package.

1. As the Google Developer Console layout might change in the future you can check their official guide on setting up leaderboards: Adding Leaderboards.

2. As the Google Developer Console layout might change in the future you can check their official guide on setting up achievements: Adding Achievements.

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 Google Play Services extension to its updated version. The first thing to notice is that Google Play Services used to be part of the runtime and all the included functions were named "**achievement_***"; these have been renamed to "**GooglePlayServices_***" 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 a GooglePlayServices GMS project is signing in to a Google Play 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):

- **GooglePlayServices_IsSignedIn()** ⇔ achievement_login_status()
- **GooglePlayServices_StartSignInIntent()** ⇔ achievement_login()
    - triggers the event of type "GooglePlayServices_SignIn" (more details)
- **GooglePlayServices_GetAccount()** gets information about the current account.
- **GooglePlayServices_SignOut()** ⇔ achievement_logout()
    - triggers the event of type "GooglePlayServices_SignOut" (more details)
- **GooglePlayServices_RevokeAccess()** revokes the rights to automatically login at application start.
    - triggers the event of type "GooglePlayServices_RevokeAccess" (more details)

[Old Version]

```
// In the previous version you would check the achievement status and if not
// signed in, you were required to login.
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 with the constant value **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 Google Play account.
if (!GooglePlayServices_IsSignedIn()) {
      GooglePlayServices_StartSignInIntent();
}
```

This function will then trigger an ASYNC SOCIAL EVENT callback, where the async_load map would contain a “type” key with the value “GooglePlayServices_SignIn” 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 account information, you can do so
// by using the method 'GooglePlayServices_GetAccount'.
var accountInfoJSON = GooglePlayServices_GetAccount();
var accountInfo = json_parse(accountInfoJSON);
```

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

Leaderboards

One of the features provided by the Google Play Services API is the creation and interaction with leaderboards. Leaderboards can be created using the Google Development Console (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):

- **GooglePlayServices_Leaderboard_SubmitScore(...)** ⇔ achievement_post_score()
  - triggers the event of type "GooglePlayServices_Leaderboard_SubmitScore" (more details)
- **GooglePlayServices_Leaderboard_ShowAll()** ⇔ achievement_show_leaderboards()
- **GooglePlayServices_Leaderboard_Show(...)** 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 with the constant value **achievement_post_score_result** and 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:
GooglePlayServices_Leaderboard_SubmitScore(global.leaderboardId, score, tag);
```

This function gives the developer the capability of adding an extra description tag to the score being submitted. This function works asynchronously and triggers an ASYNC SOCIAL EVENT callback upon completion with submission result information (GooglePlayServices_Leaderboard_SubmitScore).

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

// Show a single leaderboard
GooglePlayServices_Leaderboard_Show(global.leaderboardId);
```

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

Achievements

One of the features provided by the Google Play Services API is the creation and interaction with achievements. Achievements can be created using the Google Development Console (see Setup) and you can see and edit 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):

- **GooglePlayServices_Achievements_Show()** ⇔ achievement_show_achievements()
  - triggers the event of type "GooglePlayServices_Achievement_Show" (more details)
- **GooglePlayServices_Achievements_Increment(...)** ⇔ achievement_increment(...)
  - triggers the event of type "GooglePlayServices_Achievement_Increment" (more details)
- **GooglePlayServices_Achievements_SetSteps(...)** ⇔ achievement_post(...)
  - triggers the event of type "GooglePlayServices_Achievement_SetSteps" (more details)
- **GooglePlayServices_Achievements_Reveal(...)** ⇔ achievement_reveal(...)
  - triggers the event of type "GooglePlayServices_Achievement_Reveal" (more details)
- **GooglePlayServices_Achievements_Unlock(...)** unlocks a locked achievement.
  - triggers the event of type "GooglePlayServices_Achievement_Unlock" (more details)

[Old Version]

In previous versions you could reveal, increment, post and show in-game achievements using the code sample 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 this 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]

The new version has added functionality that allows for **unlocking** the achievement. This is the same as setting the completion to 100% (if the achievement is hidden it will be automatically revealed).

```
// First of all we need to successfully sign-in to user's the Google Play
// account and after that:
GooglePlayServices_Achievements_Reveal(achievementId);
GooglePlayServices_Achievements_SetSteps(achievementId, 75);
GooglePlayServices_Achievements_Unlock(achievementId);

GooglePlayServices_Achievements_Increment(incrementalAchievementId, amount);
GooglePlayServices_Achievements_Show();
```

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 task completion (more details).

Save Games [NEW]

This functionality suffered major changes and as such there is no migration guide for it, although 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 new feature will only work if google cloud saving is enabled under: **Game Options → Android → Social → Enable Google Cloud Saving**, otherwise none of the functionality below will work.

[Objects]

1. Each component used in the sample is represented by an object (stored inside the folder: **GooglePlayServices → Objects → SavedGames**)
2. The object **Obj_GooglePlayServices_SavedGames** is where the majority of ASYNC SOCIAL EVENT callbacks are handled.
3. Both **Obj_GooglePlayServices_DataObj** and **Obj_GooglePlayServices_Icon** are used as placeholder representations of game data to be saved.
4. The object **Obj_GooblePlayServices_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 creation of the new save slot (GooglePlayServices_SavedGames_CommitNew) is inside the callback controller object **Obj_GooglePlayServices_SavedGames**.
2. After creating the new slot we **load** (GooglePlayServices_SavedGames_Load) the slot data from Google Play servers, note that **force loading** might not be necessary since the API uses local cache to spare server calls.
3. The newly created slot will now appear on screen. If you click it, we will trigger the **opening** (GooglePlayServices_SavedGames_Open) of the slot and enter its edit mode.
4. In the edit mode we can move objects around and tap the icon to change it. After editing we can **discard** (GooglePlayServices_SavedGames_DiscardAndClose) or **commit** the changes (GooglePlayServices_SavedGames_CommitAndClose); optionally we can also **delete** the slot (GooglePlayServices_SavedGames_Delete).
5. After leaving the slot edit mode we can use the button **Show Save Slots** to call the Google Play screen overlay (GooglePlayServices_SavedGames_ShowSavedGamesUI) providing information regarding all the save slots.

Screen Recording **[NEW]**

This is a new functionality and as such there is no migration guide for it, although you can experiment with it using the provided demo sample through the in-game option: **Main Menu** → **Screen Recording**. (GooglePlayServices_VideoRecording_ShowVideoOverlay)

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 GooglePlayServices related callbacks start with "*GooglePlayServices_*".

General Functions

`GooglePlayServices_StartSignInIntent()`

Description: This function starts the intent of signing in to the Google Play account. The user is required to be signed in for all the other Google Play functionality to be usable. The function itself will not sign-in immediately, instead it will trigger an ASYNC SOCIAL EVENT that can be listened to in order to check if the sign-in process succeeded.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SignIn"

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

`GooglePlayServices_IsSignedIn()`

Description: This function returns whether or not the user is currently logged into his/her Google Play account.

Returns: {*bool*} 1 - is logged in; 0 - is not logged in

`GooglePlayServices_SignOut()`

Description: This function will temporarily make the application sign-out of the user's Google Play account. It does so "temporarily" because the application automatically signs the user in on re-opening the app.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SignOut"

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

`GooglePlayServices_RevokeAccess()`

Description: This function signs the user out and revokes the access to the user's login. Revoking means that the application will not automatically login again upon re-opening. The application will need to call the 'GooglePlayServices_StartSignInIntent' again in order for the login process to be initialised. This event triggers an ASYNC SOCIAL EVENT with a type equal to its name.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_RevokeAccess"

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

`GooglePlayServices_GetAccount()`

Description: This function queries the Google Play server for information on the current account and returns a json formatted string containing account information (see AccountJSON)

Returns: {*string*} json formatted string that can be passed into the **json_parse** method.

Player Functions

`GooglePlayServices_Player_Current()`

Description: This function queries the Google Play server for information on the current signed-in player. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Player_Current"

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

player: {PlayerJSON} The information of the player in a json formatted string that can be passed into the **json_parse** method.

`GooglePlayServices_Player_CurrentID()`

Description: This function queries the Google Play server for information on the current player's unique id. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Player_CurrentID"

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

playerId: {*string*} The current player's unique id.

Achievements Functions

`GooglePlayServices_Achievements_Show()`
Description: This function will call the Google Play Services achievement overlay with all your achievement information.

Returns: N/A

`GooglePlayServices_Achievements_SetSteps(achievementId, steps)`
Description: This function requests the Google Play Services API to set the achievement progress to a given amount of steps. Incremental achievements require a specific amount of steps before they are set as complete. (see GooglePlayServices_Achievements_SetSteps). The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

{string} **achievementId**: the unique identifier of the achievement.

{*real*} **steps**: the amount of steps to set to.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Achievements_SetSteps"

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

`GooglePlayServices_Achievements_Increment(achievementId, steps)`
Description: This function requests the Google Play Services API to increment the achievement progress by a given amount of steps. Incremental achievements require a specific amount of steps before they are set as complete. (see GooglePlayServices_Achievements_SetSteps). The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

{string} **achievementId**: the unique identifier of the achievement.

{*real*} **steps**: the amount of steps to increment by.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Achievements_Increment"

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

`GooglePlayServices_Achievements_Reveal(achievementId)`

Description: This function requests the Google Play Services API to change the state of a given achievement to 'revealed' for the currently signed in player. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

{string} **achievementId**: the unique identifier of the achievement.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Achievements_Reveal"

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

`GooglePlayServices_Achievements_Unlock(achievementId)`

Description: This function requests the Google Play Services API to unlock the given achievement for the currently signed in player. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

{string} **achievementId**: the unique identifier of the achievement.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Achievements_Unlock"

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

Leaderboard Functions

`GooglePlayServices_Leaderboard_SubmitScore(leaderId, score, scoreTag)`
Description: This function requests the Google Play Services API to submit a score to the given leaderboard. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

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).

{string} **tag**: a tag that will be added to the value being submitted to the leaderboard. (note that this value is **required**, if you don't want to set a tag use an empty string).

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_Achievements_SubmitScore"

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

leaderboardId: {*string*} The leaderboard the score was submitted to.

score: {real} The submitted score

scoreTag: {*string*} The tag for the current submission

report: {LeaderboardReportJSON} The submit information in a json formatted string that can be passed into the **json_parse** method.

`GooglePlayServices_Leaderboard_Show(leaderId)`
Description: This function will call the Google Play Services overlay for a specific leaderboard.

Params:

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

Returns: N/A

`GooglePlayServices_Leaderboard_ShowAll()`
Description: This function will call the general Google Play Services leaderboards overlay. Here the user will have access to all the existing leaderboards of the current application.

Returns: N/A

Save Game Functions

These functions require Google Cloud saving permissions and will only work if the option is enabled under: **Game Options → Android → Social → Enable Google Cloud Saving**, otherwise none of the functionality below will work.

`GooglePlayServices_SavedGames_ShowSavedGamesUI(title, add, delete, max)`

Description: This function requests Google Play Services API to open the saved games UI overlay, giving you the ability to see, add (optional) and delete (optional) the save slots.

Params:

{string} **title**: The text to be used as the title of the overlay.

{boolean} **add**: Whether or not the 'Add' slot button should be displayed.

{boolean} **delete**: Whether or not the 'Delete' slot button should be displayed.

{real} **max**: Sets the maximum amount of slots allowed. This can be useful to limit the number of saves you want the player to be able to create.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

<selecting a slot>

type: "GooglePlayServices_SavedGames_ShowSavedGamesUI_OnOpen"

snapshotMetadata: (json formatted string - SnapshotMetadataJSON)

<creating a new slot>

type: "GooglePlayServices_SavedGames_ShowSavedGamesUI_OnNew"

<closing the overlay>

type: "GooglePlayServices_SavedGames_ShowSavedGamesUI_OnClose"

`GooglePlayServices_SavedGames_Open(name)`

Description: This function requests the Google Play Services API to open a given save slot given its unique name identifier. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

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

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SavedGames_Open"

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

snapshotMetadata: {SnapshotMetadataJSON} a json formatted string of the snapshot metadata that can be parsed using **json_parse**.

data: (the string previously saved to the Google Play Services)

`GooglePlayServices_SavedGames_CommitAndClose(name, desc, data, coverPath)`

Description: This function requests the Google Play Services API to commit data to a given save slot. The unique identifier of the slot needs to exist meaning that this function will only update already existing files (see GooglePlayServices_SavedGames_CommitNew for creating new ones) but the 'description', 'data' and 'coverPath' can be changed if needed. This function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

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

{string} **description**: the description of the current save slot.

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

{string} **coverPath**: a string with the path to the image to be used as a save slot cover image.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SavedGames_CommitAndClose"

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

`GooglePlayServices_SavedGames_DiscardAndClose(name)`

Description: This function requests the Google Play Services API to discard changes and close the currently opened save slot.

Params:

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

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SavedGames_DiscardAndClose"

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

`GooglePlayServices_SavedGames_CommitNew(name, desc, data, coverPath)`

Description: This function requests the Google Play Services API to commit data to a given save slot. This function will create a new slot, if you want to update an already existing slot use the GooglePlayServices_SavedGames_CommitAndClose function. This function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

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

{string} **description**: the description of the current save slot.

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

{string} **coverPath**: a string with the path to the image to be used as a save slot cover image.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SavedGames_CommitNew"

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

snapshotMetadata: {SnapshotMetadataJSON} a json formatted string of the snapshot metadata that can be parsed using **json_parse**.

`GooglePlayServices_SavedGames_Delete(name)`

Description: This function requests the Google Play Services API to delete a given save slot given its unique identifier. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

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

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SavedGames_Delete"

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

`GooglePlayServices_SavedGames_Load(forceReload)`

Description: This function requests the Google Play Services API to query all the save slot metadata of the currently signed-in user. The function will not return any value but it will create a request that will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

{boolean} **forceReload**: Whether or not the current local cache should be cleared and a new fetch should be performed from the server.

Returns: N/A

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_SavedGames_Load"

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

snapshots: {SnapshotMetadataJSON[ ]} a json formatted string of an **array** with all the existing snapshots metadata that can be parsed using **json_parse**.

Screen Recording Functions

`GooglePlayServices_VideoRecording_ShowVideoOverlay()`

Description: This method will enable the video recording screen overlay. With this feature the user will be able to record their screen and save the recordings to their phone's memory. Front camera and microphone can also be recorded.

Returns: N/A

Utility Functions

`GooglePlayServices_SetClientID(clientId)`

Description: This function should be called **before** signing in, and is useful for Google/Firebase authentication, should it be needed.

Params:

{string} **clientId**: The unique client id string to be used.

Returns: N/A

`GooglePlayServices_GetServerAuthCode()`

Description: This function returns the server AuthCode, useful for authentication.

Returns: {*string*}

`GooglePlayServices_UriToPath(uri)`

Description: Some of the function callbacks above return a URI (unique resource identifier). However if the user needs to open these same files it is necessary to convert these URIs into paths. This function requests the Google Play Services API for the path to a given URI. The function will not return the path immediately but instead will return a request id and will trigger an ASYNC SOCIAL EVENT callback when it is ready.

Params:

{string} **uri**: The uri to get the path from.

Returns: {real} The id of the request.

Triggers: ASYNC SOCIAL EVENT

type: "GooglePlayServices_UriToPath"

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

ind: {*real*} the id of the request this callback refers to.

path: {*string*} the path to the resource.

JSON Structs

AccountJSON

→ displayName
→ email
→ familyName
→ givenName
→ IdToken
→ photoUrl
→ serverAuthCode

PlayerJSON

→ bannerImageLandscapeUri
→ bannerImagePortraitUri
→ displayName
→ hiResImageUri
→ iconImageUri
→ lastPlayedWithTimestamp
→ currentXpTotal
→ lastLevelUpTimestamp
→ currentLevelNumber
→ currentMaxXp
→ currentMinXp
→ nextLevelNumber
→ nextMaxXp
→ nextMinXp
→ playerId
→ retrievedTimestamp
→ title
→ hasHiResImage
→ hasIconImage

GameJSON

→ areSnapshotsEnabled
→ achievementTotalCount
→ applicationId
→ description
→ developerName
→ displayName

→ featuredImageUri
→ hiResImageUri
→ iconImageUri
→ leaderboardCount
→ primaryCategory
→ secondaryCategory
→ themeColor
→ gamepadSupport

SnapshotMetadataJSON

→ coverImageAspectRatio
→ coverImageUri
→ description
→ deviceName
→ game (GameJSON)
→ lastModifiedTimestamp
→ owner (PlayerJSON)
→ playedTime
→ progressValue
→ uniqueName
→ hasChangePending

LeaderboardReportJSON

→ daily:
    → score
    → scoreTag
    → isNewBest
→ weekly:
    → score
    → scoreTag
    → isNewBest
→ allTime:
    → score
    → scoreTag
    → isNewBest