luasteam Documentation

USPGameDev

Feb 07, 2019

Contents:

luasteam enables you to use SteamWorks API from Lua, for example (but not limited to) when you're building games with Love2D.

To learn how to use luasteam, check our Getting Started section.

While using this documentation, you may also like to check the SteamWorks API Reference and SteamWorks API Overview.

CHAPTER 1

Getting started

1.1 Installation

To install luasteam, you can use one of our pre-compiled binaries in our releases page. Make sure you're using the same version of this documentation as of the release you downloaded.

Download the correct binary for your platform and rename it to simply `luasteam.ext`. For example, if you're using Windows 32-bits, download `win32_luasteam.dll` and rename it to `luasteam.dll`. Then copy this file to the same directory as your lua files, and make sure `require 'luasteam'` works.

Warning: You also need to copy the correct SteamWorks library to the same directory. This library is not on the luasteam repository, and must be downloaded through the SteamWorks website. This version of luasteam is guaranteed to work with SteamWorks SDK v1.42, but probably works with future versions.

After you download the SDK, you should copy the corresponding library for your platform. Here is a cheat-sheet:

- Linux 32: `sdk/redistributable_bin/linux32/libsteam_api.so`
- Linux 64: `sdk/redistributable_bin/linux64/libsteam_api.so`
- Windows 32: `sdk/redistributable_bin/steam_api.dll`
- Windows 64: `sdk/redistributable_bin/win64/steam_api64.dll`
- OSX: `sdk/redistributable_bin/osx32/libsteam_api.dylib`

1.2 Basic Usage

luasteam tries to be very similar to the SteamWorks API, that way you don't need to learn two different APIs. Big differences (other than lua/C++ differences) are documented.

1.2.1 Initialization and Shutdown

Using the usual SteamWorks API, your code will look like:

```
#include <steam_api.h>

SteamAPI_Init();
// ...
// when game is closing
SteamAPI_Shudown()
```

Doing this in luasteam should be as easy as:

```
local Steam = require 'luasteam'

Steam.init()
-- ...
-- when game is closing
Steam.shutdown()
```

Check the overview for more info on initialization and shutdown. When developing your game, remember to have Steam turned on and use a `steam_appid.txt` file.

1.2.2 Normal functions

The lua bindings for normal functions are very similar to the C++ API, so for example

```
SteamFriends()->ActivateGameOverlay("achievements");
```

becomes

```
Steam.friends.activateGameOverlay("achievements")
```

1.2.3 Callbacks

Callbacks work a little different, for example, the `GameOverlayActivated_t` callback in `ISteamUserFriends` can be used creating a function named `onGameOverlayActivated` inside `userStats`.

Original code:

```
class Listener {
    STEAM_CALLBACK(Listener, OnGameOverlayActivated, GameOverlayActivated_t);
};

void Listener::OnGameOverlayActivated(GameOverlayActivated_t* data) {
    if (data->m_bActive)
        printf("Steam overlay now active\n" );
    else
        printf("Steam overlay now inactive\n");
}
```

Code using luasteam:

```
function Steam.userStats.onGameOverlayActivated(data)
    if data.active then
        print("Steam overlay now active")
    else
        print("Steam overlay now inactive")
    end
end
```

1.2.4 CallResults

Using CallResults is also slightly different. Instead of receiving a `SteamAPICall_t` and registering to listen to it, you simply pass a function as the last argument to the function. This function receives two arguments: `data` is the object returned by the CallResult and `err` is a boolean indicating if there was an IO error.

Original code:

```
class Listener {
public:
    void FindTestLeaderboard(const char *name);

private:
    void OnLeaderboardFindResult(LeaderboardFindResult_t *data, bool err);
    CCallResult<Listener, LeaderboardFindResult_t> leaderboardFindResult;
};

void Listener::OnLeaderboardFindResult(LeaderboardFindResult_t *data, bool err ) {
    if (err || data->m_bLeaderboardFound == 0)
        printf("Leaderboard not found!\n");
    else
        printf("Leaderboard found!\n");
}

// Make the request
void Listener::FindTestLeaderboard() {
    SteamAPICall_t call = SteamUserStats()->FindLeaderboard("test");
    leaderboardFindResult.Set(call, this, &Listener::OnLeaderboardFindResult);
}
```

Code in luasteam

```
Steam.userStats.findLeaderboard("test", function(data, err)
    if err or not data.leaderboardFound then
        print("Leaderboard not found!")
    else
        print("Leaderboard found!")
    end
end)
```

Warning: To use Callbacks and Call Results, you **must** constantly call `Steam.runCallbacks()`, preferably in your game loop.

1.2.5 64-bit integers

Some identifiers in the SteamWorks API are 64-bit integers (for example, SteamID, Leaderboard Handle, etc.). In this documentation, these use *uint64* types instead of *number*.

Since Lua 5.1 does not support integers, and doubles (the default number type) can't hold a 64-bit integer with no error, we use userdata to keep such integers (even in Lua versions that support integers).

They can only be compared for equality or converted to strings (using the *tostring* function), since doing any math on them doesn't make any sense. You can use *extra.parseUint64()* to parse them from strings.

```
local original = Steam.user.getSteamID()
local str = tostring(original)
print("Your id is " .. str)
local id = Steam.extra.parseUint64(str)
-- equality works, even though they are different userdata instances
assert(id == original)
```

CHAPTER 2

SteamAPI Core Functions

2.1 List of Functions

2.2 Function Reference

init()

Returns

(*boolean*) **true** indicates that all required interfaces have been acquired and are accessible. **false** indicates one of the following conditions:

- The Steam client isn't running. A running Steam client is required to provide implementations of the various Steamworks interfaces.
- The Steam client couldn't determine the App ID of game. If you're running your application from the executable or debugger directly then you must have a `steam_appid.txt` in your game directory next to the executable, with your app ID in it and nothing else. Steam will look for this file in the current working directory. If you are running your executable from a different directory you may need to relocate the `steam_appid.txt` file.
- Your application is not running under the same OS user context as the Steam client, such as a different user or administration access level.
- Ensure that you own a license for the App ID on the currently active Steam account. Your game must show up in your Steam library.
- Your App ID is not completely set up, i.e. in `Release State: Unavailable`, or it's missing default packages.

SteamWorks SteamAPI_Init

Initializes the Steamworks API. **Must** be the first thing you call after loading the library, do it before anything else in your game.

See Initialization and Shutdown for additional information.

Example:

```
local Steam = require 'luasteam'

if not Steam.init() then
    error("Steam couldn't initialize")
end
```

shutdown()

Returns nothing

SteamWorks SteamAPI_Shutdown

Shuts down the Steamworks API, releases pointers and frees memory.

You should call this during process shutdown if possible.

This will not unhook the Steam Overlay from your game as there's no guarantee that your rendering API is done using it.

Example:

```
function onMyGameClosing()
    Steam.shutdown()
end
```

runCallbacks()

Returns nothing

SteamWorks SteamAPI_RunCallbacks

Dispatches callbacks and call results to all of the registered listeners.

It's best to call this at >10Hz, the more time between calls, the more potential latency between receiving events or results from the Steamworks API. Most games call this once per render-frame. All registered listener functions will be invoked during this call, in the callers thread context.

runCallbacks() is safe to call from multiple threads simultaneously, but if you choose to do this, callback code could be executed on any thread. One alternative is to call *runCallbacks()* from the main thread only, and call releaseCurrentThreadMemory() **(missing)** regularly on other threads.

Example:

```
function myGameLoop(dt)
    Steam.runCallbacks()
end
```

CHAPTER 3

ISteamFriends

3.1 List of Functions

- *friends.activateGameOverlay()*
- *friends.activateGameOverlayToWebPage()*
- *friends.getFriendPersonaName()*

3.2 List of Callbacks

- *friends.onGameOverlayActivated()*

3.3 Function Reference

`friends.`**`activateGameOverlay`**(*dialog*)

Parameters **`dialog`** (*string*) – The dialog to open. Valid options are: “friends”, “community”, “players”, “settings”, “officialgamegroup”, “stats”, “achievements”.

Returns nothing

SteamWorks ActivateGameOverlay

Activates the Steam Overlay to a specific dialog.

Example:

```
Steam.friends.activateGameOverlay('stats')
```

`friends.`**`activateGameOverlayToWebPage`**(*url*)

Parameters **url** (*string*) – The webpage to open. (A fully qualified address with the protocol is required, e.g. "http://www.steampowered.com")

Returns nothing

SteamWorks ActivateGameOverlayToWebPage

Activates Steam Overlay web browser directly to the specified URL.

Example:

```
Steam.friends.activateGameOverlayToWebPage('https://www.google.com')
```

friends.**getFriendPersonaName**(*steam_id*)

Parameters **steam_id** (*uint64*) – The Steam ID of the other user.

Returns (*string*) The current users persona name. Returns an empty string (""), or "[unknown]" if the Steam ID is invalid or not known to the caller.

SteamWorks GetFriendPersonaName

Gets the specified user's persona (display) name.

This will only be known to the current user if the other user is in their friends list, on the same game server, in a chat room or lobby, or in a small Steam group with the local user.

Example:

```
steam_id = getSteamIdSomehow()
print("Friend's name is:", Steam.friends.getFriendPersonaName(steam_id))
```

3.4 Callbacks Reference

Warning: Remember callbacks are functions that you should override in order to receive the events, and not call directly.

Also, you **must** constantly call `Steam.runCallbacks()` (preferably in your game loop) in order for your callbacks to be called.

friends.**onGameOverlayActivated**(*data*)

Parameters **data** (*table*) – A table with a single

Returns nothing

SteamWorks GameOverlayActivated_t

Posted when the Steam Overlay activates or deactivates. The game can use this to be pause or resume single player games.

Example:

```
function Steam.friends.onGameOverlayActivated(data)
    print('Overlay active is', data.active)
end
```

CHAPTER 4

ISteamUser

4.1 List of Functions

- *user.getPlayerSteamLevel()*
- *user.getSteamID()*

4.2 Function Reference

user.**getPlayerSteamLevel**()

Returns (*number*) The level of the current user.

SteamWorks GetPlayerSteamLevel

Gets the Steam level of the user, as shown on their Steam community profile.

Example:

```
print('Let me show you some magic')
print('Your Steam Level is...')
print(Steam.user.getPlayerSteamLevel() .. '!!!')
```

user.**getSteamID**()

Returns (*uint64*) The SteamID of the current user.

SteamWorks GetSteamID

Gets the Steam ID of the account currently logged into the Steam client. This is commonly called the 'current user', or 'local user'.

A Steam ID is a unique identifier for a Steam accounts, Steam groups, Lobbies and Chat rooms, and used to differentiate users in all parts of the Steamworks API.

Example:

```lua
local my_id = Steam.user.getSteamID()
function isSteamIDFromUser(steam_id)
    return steam_id == my_id
end
```

CHAPTER 5

ISteamUserStats

5.1 List of Functions

- *userStats.getAchievement()*
- *userStats.setAchievement()*
- *userStats.resetAllStats()*
- *userStats.storeStats()*
- *userStats.requestCurrentStats()*
- *userStats.findLeaderboard()*
- *userStats.findOrCreateLeaderboard()*
- *userStats.getLeaderboardEntryCount()*
- *userStats.getLeaderboardName()*
- *userStats.getLeaderboardSortMethod()*
- *userStats.getLeaderboardDisplayType()*
- *userStats.uploadLeaderboardScore()*
- *userStats.downloadLeaderboardEntries()*

5.2 Function Reference

userStats.**getAchievement**(*name*)

Parameters **name** (*string*) – The 'API Name' of the achievement.

Returns

(*boolean*) This function returns true upon success if all of the following conditions are met; otherwise, false.

- *userStats.requestCurrentStats()* has completed and successfully returned its callback.
- The 'API Name' of the specified achievement exists in App Admin on the Steamworks website, and the changes are published.

Returns (*boolean?*) If the call is successful, returns a second value, the unlock status of the achievement.

SteamWorks GetAchievement

Gets the unlock status of the Achievement.

The equivalent function for other users is `userStats.getUserAchievement()` **(missing)**.

Example:

```
local success, achieved = Steam.userStats.getAchievement('ach_name')
if success and achieved then
    print('Yep, you have the achievement')
end
```

`userStats.`**`setAchievement`**`(`*name*`)`

Parameters **`name`** (*string*) – The 'API Name' of the Achievement to unlock.

Returns

(*boolean*) This function returns true upon success if all of the following conditions are met; otherwise, false.

- The specified achievement 'API Name' exists in App Admin on the Steamworks website, and the changes are published.
- *userStats.requestCurrentStats()* has completed and successfully returned its callback.

SteamWorks SetAchievement

Unlocks an achievement.

You must have called *userStats.requestCurrentStats()* and it needs to return successfully via its callback prior to calling this!

You can unlock an achievement multiple times so you don't need to worry about only setting achievements that aren't already set. This call only modifies Steam's in-memory state so it is quite cheap. To send the unlock status to the server and to trigger the Steam overlay notification you must call *userStats.storeStats()*.

Example:

```
if achievementConditionSatisfied() and doesntHaveAchievement() then
    Steam.userStats.setAchievement('ach_name')
    Steam.userStats.storeStats() -- shows overlay notification
end
```

`userStats.`**`resetAllStats`**`(`*achievementsToo*`)`

Parameters **`achievementsToo`** (*boolean*) – Also reset the user's achievements?

Returns (*boolean*) true indicating success if *userStats.requestCurrentStats()* has already been called and successfully returned its callback; otherwise false.

SteamWorks ResetAllStats

Resets the current users stats and, optionally achievements.

This automatically calls *userStats.storeStats()* to persist the changes to the server. This should typically only be used for testing purposes during development. Ensure that you sync up your stats with the new default values provided by Steam after calling this by calling *userStats.requestCurrentStats()*.

Example:

```
if dev_mode and keypressed('f10') then
    Steam.userStats.resetAllStats(true)
end
```

userStats.**storeStats**()

Returns

(*boolen*) This function returns true upon success if all of the following conditions are met; otherwise, false.

- *userStats.requestCurrentStats()* has completed and successfully returned its callback.
- The current game has stats associated with it in the Steamworks Partner backend, and those stats are published.

SteamWorks StoreStats

Send the changed stats and achievements data to the server for permanent storage.

If this fails then nothing is sent to the server. It's advisable to keep trying until the call is successful.

This call can be rate limited. Call frequency should be on the order of minutes, rather than seconds. You should only be calling this during major state changes such as the end of a round, the map changing, or the user leaving a server. This call is required to display the achievement unlock notification dialog though, so if you have called *userStats.setAchievement()* then it's advisable to call this soon after that.

If you have stats or achievements that you have saved locally but haven't uploaded with this function when your application process ends then this function will automatically be called.

You can find additional debug information written to the `%steam_install%\logs\stats_log.txt` file.

If the call is successful you will receive a `userStats.userStatsStored()` callback. If **result** has a result of **"InvalidParam"**, then one or more stats uploaded has been rejected, either because they broke constraints or were out of date. In this case the server sends back updated values and the stats should be updated locally to keep in sync.

If one or more achievements has been unlocked then this will also trigger a `userStats.userAchievementStored()` callback.

Example:

```
function onMatchEnd()
    Steam.userStats.storeStats()
end
```

userStats.**requestCurrentStats**()

Returns (*boolean*) Only returns false if there is no user logged in; otherwise, true.

SteamWorks RequestCurrentStats

Asynchronously request the user's current stats and achievements from the server.

You must **always call this first** to get the initial status of stats and achievements. Only after the resulting callback comes back can you start calling the rest of the stats and achievement functions for the current user.

The equivalent function for other users is `userStats.requestUserStats()` **(missing)**.

Triggers a `userStats.userStatsReceived()` callback.

Example:

```
-- before any achievement/stats stuff
Steam.userStats.requestCurrentStats()

function Steam.userStats.userStatsReceived()
    can_do_stats_stuff = true
end
```

`userStats.`**`findLeaderboard`**(*name*, *callback*)

Parameters

- **`name`** (*`string`*) – The name of the leaderboard to find. Must not be longer than 128 bytes.
- **`callback`** (*`function`*) – Called asynchronously when this function returns. See below.

Returns nothing

SteamWorks FindLeaderboard

Gets a leaderboard by name.

You must call either this or *`userStats.findOrCreateLeaderboard()`* to obtain the leaderboard handle which is valid for the game session for each leaderboard you wish to access prior to calling any other Leaderboard functions.

callback(data, err) receives two arguments:

- **data** (*table*) – Similar to LeaderboardFindResult_t, or **nil** if **err** is **true**.
  - **data.steamLeaderboard** (*uint64*) – Handle to the leaderboard that was searched for. A special value is returned if no leaderboard was found.
  - **data.leaderboardFound** (*boolean*) – Was the leaderboard found? **true** if it was, **false** if it wasn't.
- **err** (*boolean*): **true** if there was any IO error with the request.

Example:

```
Steam.userStats.findLeaderboard('l_name', function(data, err)
    if err or not data.leaderboardFound then
        print('Something happened')
    elseif
        uploadScoresHelper(data.steamLeaderboard)
    end
end)
```

`userStats.`**`findOrCreateLeaderboard`**(*name*, *sortMethod*, *displayType*, *callback*)

Parameters

- **`name`** (*`string`*) – The name of the leaderboard to find or create. Must not be larger than 128 bytes.

- **sortMethod** (*string*) – The sort order of the new leaderboard if it's created. Must be 'Ascending' or 'Descending' (see ELeaderboardSortMethod).
- **displayType** (*string*) – The display type (used by the Steam Community web site) of the new leaderboard if it's created. Must be one of: 'Numeric', 'TimeSeconds' or 'TimeMilliSeconds' (see ELeaderboardDisplayType).
- **callback** (*function*) – Called asynchronously when this function returns. It must be of the same type as the callback in *userStats.findLeaderboard()*.

Returns nothing

SteamWorks FindOrCreateLeaderboard

Gets a leaderboard by name, it will create it if it's not yet created.

You must call either this or *userStats.findLeaderboard()* to obtain the leaderboard handle which is valid for the game session for each leaderboard you wish to access prior to calling any other Leaderboard functions.

Leaderboards created with this function will not automatically show up in the Steam Community. You must manually set the Community Name field in the App Admin panel of the Steamworks website. As such it's generally recommended to prefer creating the leaderboards in the App Admin panel on the Steamworks website and using *userStats.findLeaderboard()* unless you're expected to have a large amount of dynamically created leaderboards.

Example:

```
Steam.userStats.findOrCreateLeaderboard('l_name', 'Ascending', 'Numeric',␣
↪function(data, err)
    if err or not data.leaderboardFound then
        print('Something happened')
    elseif
        uploadScoresHelper(data.steamLeaderboard)
    end
end)
```

userStats.**getLeaderboardName**(*steamLeaderboard*)

Parameters **steamLeaderboard** (*uint64*) – A leaderboard handle obtained from *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()*.

Returns (*string*) The name of the leaderboard. Returns an empty string if the leaderboard handle is invalid.

SteamWorks GetLeaderboardName

Returns the name of a leaderboard handle.

Example:

```
function printLeaderboardInfo(handle)
    print('Leaderboard name: ' .. Steam.userStats.getLeaderboardName(handle))
    print('Entries: ' .. Steam.userStats.getLeaderboardEntryCount(handle))
    print('Sort Method: ' .. Steam.userStats.getLeaderboardSortMethod(handle))
    print('Display Type: ' .. Steam.userStats.getLeaderboardDisplayType(handle))
end
```

userStats.**getLeaderboardEntryCount**(*steamLeaderboard*)

Parameters **steamLeaderboard** (*uint64*) – A leaderboard handle obtained from *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()*.

Returns (*number*) The number of entries in the leaderboard. Returns 0 if the leaderboard handle is invalid.

SteamWorks GetLeaderboardEntryCount

Returns the total number of entries in a leaderboard.

This is cached on a per leaderboard basis upon the first call to *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()* and is refreshed on each successful call to *userStats.downloadLeaderboardEntries()*, `userStats.downloadLeaderboardEntriesForUsers()` **(missing)**, and *userStats.uploadLeaderboardScore()*.

Example: See *userStats.getLeaderboardName()*'s example.

`userStats.`**`getLeaderboardSortMethod`**(*steamLeaderboard*)

Parameters **steamLeaderboard** (*uint64*) – A leaderboard handle obtained from *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()*.

Returns (*string?*) The sort method of the leaderboard, either "Ascending" or "Descending". Returns **nil** if the leaderboard handle is invalid.

SteamWorks GetLeaderboardSortMethod

Returns the sort order of a leaderboard handle.

Example: See *userStats.getLeaderboardName()*'s example.

`userStats.`**`getLeaderboardDisplayType`**(*steamLeaderboard*)

Parameters **steamLeaderboard** (*uint64*) – A leaderboard handle obtained from *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()*.

Returns (*string?*) The display type of the leaderboard, one of "Numeric", "TimeSeconds" or "TimeMilliSeconds". Returns **nil** if the leaderboard handle is invalid.

SteamWorks GetLeaderboardDisplayType

Returns the display type of a leaderboard handle.

Example: See *userStats.getLeaderboardName()*'s example.

`userStats.`**`uploadLeaderboardScore`**(*steamLeaderboard*, *uploadScoreMethod*, *score*, *details*, *callback*)

Parameters

- **steamLeaderboard** (*uint64*) – A leaderboard handle obtained from *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()*.
- **uploadScoreMethod** (*string*) – Do you want to force the score to change, or keep the previous score if it was better? Either "KeepBest" or "ForceUpdate".
- **score** (*number*) – The score to upload. Must be a 32-bit integer.
- **details** (*string?*) – Optional string with details surrounding the unlocking of this score. Size must be a multiple of four, and at most 256 bytes. Will be converted to an array of 32-bit integers.

- **callback** (*function*) – Called asynchronously when this function returns. See below.

Returns nothing

SteamWorks UploadLeaderboardScore

Uploads a user score to a specified leaderboard.

Details are optional game-defined information which outlines how the user got that score. For example if it's a racing style time based leaderboard you could store the timestamps when the player hits each checkpoint. If you have collectibles along the way you could use bit fields as booleans to store the items the player picked up in the playthrough.

Uploading scores to Steam is rate limited to 10 uploads per 10 minutes and you may only have one outstanding call to this function at a time.

callback(data, err) receives two arguments:

- **data** (*table*) – Similar to LeaderboardScoreUploaded_t, or **nil** if there was **err** is **true**.
  - **data.success** (*boolean*) – Was the call successful? Returns **true** if the call was successful, **false** on failure, for example:
    - The amount of details sent exceeds 256 bytes.
    - The leaderboard is set to "Trusted" in App Admin on Steamworks website, and will only accept scores sent from the Steam Web API.
  - **data.steamLeaderboard** (*uint64*) – Handle to the leaderboard that was searched for. A special value is returned if no leaderboard was found.
  - **data.score** (*number*) – The score that was attempted to set.
  - **data.scoreChanged** (*boolean*) – **true** if the score on the leaderboard changed otherwise **false** if the existing score was better.
  - **data.globalRankNew** (*number*) – The new global rank of the user on this leaderboard.
  - **data.globalRankPrevious** (*number*) – The previous global rank of the user on this leaderboard; 0 if the user had no existing entry in the leaderboard.
- **err** (*boolean*): **true** if there was any IO error with the request.

Example:

```
function uploadScoresHelper(handle)
    local score = getScore()
    Steam.userStats.uploadLeaderboardScore(handle, "KeepBest", score, nil,␣
↪function(data, err)
        if err or not data.success then
            print('Upload score failed')
        else
            print('Upload score success. New rank is: ' .. data.globalRankNew)
        end
    end)
end
```

`userStats.`**`downloadLeaderboardEntries`**(*steamLeaderboard*, *dataRequest*, *rangeStart*, *rangeEnd*, *callback*)
`userStats.`**`downloadLeaderboardEntries`**(*steamLeaderboard*, *dataRequest*, *callback*)

Parameters

- **steamLeaderboard** (*uint64*) – A leaderboard handle obtained from *userStats.findLeaderboard()* or *userStats.findOrCreateLeaderboard()*.

- **dataRequest** (*string*) – The type of data request to make. Must be one of "Global", "GlobalAroundUser" or "Friends" (see ELeaderboardDataRequest).
- **rangeStart** (*number*) – The index to start downloading entries relative to **dataRequest**. Must **not** be supplied if **dataRequest** is "Friends".
- **rangeEnd** (*number*) – The last index to retrieve entries relative to **dataRequest**. Must **not** be supplied if **dataRequest** is "Friends".
- **callback** (*function*) – Called asynchronously when this function returns. See below.

Returns nothing

SteamWorks DownloadLeaderboardEntries

Fetches a series of leaderboard entries for a specified leaderboard.

You can ask for more entries than exist, then this will return as many as do exist.

If you want to download entries for an arbitrary set of users, such as all of the users on a server then you can use `userStats.downloadLeaderboardEntriesForUsers()` **(missing)** which takes an array of Steam IDs.

callback(data, err) receives two arguments:

- **data** (*table*) – An array of tables similar to LeaderboardEntry_t, or **nil** if there was **err** is **true**.
  - **data[i].steamIDUser** (*uint64*) – User who this entry belongs to. You can use *friends.getFriendPersonaName()* and `friends.getSmallFriendAvatar()` **(missing)** to get more info.
  - **data[i].globalRank** (*number*) – The global rank of this entry ranging from [1..N], where N is the number of users with an entry in the leaderboard.
  - **data[i].score** (*number*) – The raw score as set in the leaderboard.
  - **data[i].details** (*string*) – Details of the entry. String is used as a byte array, so may contain a `'\0'` in the middle.
  - **data[i].UGC** (*uint64*) – Handle for the UGC attached to the entry. A special value if there is none.
- **err** (*boolean*): **true** if there was any IO error with the request.

Warning: This function has two major differences from the SteamWorks API.

- If the data request is "Friends", you **must not** use the **rangeStart** and **rangeEnd** parameters (see the second example).
- The callback is not called with a table similar to LeaderboardScoresDownloaded_t, and there is no need to use the function GetDownloadedLeaderboardEntry, since this is already done for you. The callback already receives a list of objects like LeaderboardEntry_t.

Examples:

```
function showGlobalEntries(handle)
    Steam.userStats.downloadLeaderboardEntries(handle, 'Global', 1, 1000,␣
↪function(data, err)
        if err then
            print('Error happened')
        else
            for _, user in ipairs(data) do
                print('Rank #' .. user.globalRank .. ': ' .. user.score)
```

(continues on next page)

(continued from previous page)

```
            end
        end
    end
end
```

```
function showFriendsEntries(handle)
    Steam.userStats.downloadLeaderboardEntries(handle, 'Friends', function(data, err)
        if err then
            print('Error happened')
        else
            for _, user in ipairs(data) do
                local name = Steam.friends.getFriendPersonaName(user.steamIDUser)
                print('Friend ' .. name .. ': ' .. user.score)
            end
        end
    end
end
```

CHAPTER 6

ISteamUtils

6.1 List of Functions

- *utils.getAppID()*

6.2 Function Reference

utils.**getAppID**()

Returns (*number*) The AppID.

SteamWorks GetAppID

Gets the App ID of the current process.

Example:

```
print("My app id is " .. Steam.utils.getAppID())
```

CHAPTER 7

ISteamUGC

7.1 List of Functions

- *UGC.createItem()*
- *UGC.startItemUpdate()*
- *UGC.setItemContent()*
- *UGC.setItemDescription()*
- *UGC.setItemPreview()*
- *UGC.setItemTitle()*
- UGC.submitItemUpdateResult()
- *UGC.getNumSubscribedItems()*
- *UGC.getSubscribedItems()*
- *UGC.getItemState()*
- *UGC.getItemInstallInfo()*
- *UGC.getItemUpdateProgress()*
- *UGC.startPlaytimeTracking()*
- *UGC.stopPlaytimeTracking()*
- *UGC.stopPlaytimeTrackingForAllItems()*

7.2 Function Reference

UGC.**createItem**(*consumerAppId*, *fileType*, *callback*)

Parameters

- **consumerAppId** (*number*) – The App ID that will be using this item.
- **fileType** (*string*) – The type of UGC to create. Must be one of 'Community', 'Microtransaction', 'Collection', 'Art', 'Video', 'Screenshot', 'WebGuide', 'IntegratedGuide', 'Merch', 'ControllerBinding', 'SteamVideo' or 'GameManagedItem' (see EWorkshopFileType).
- **callback** (*function*) – Called asynchronously when this function returns. See below.

Returns nothing

SteamWorks CreateItem

Creates a new workshop item with no content attached yet.

callback(data, err) receives two arguments:

- **data** (*table*) – Similar to CreateItemResult_t, or **nil** if **err** is **true**.
  - **data.result** (*number*) – The result of the operation. See EResult.
  - **data.publishedFileId** (*uint64*) – The new items unique ID.
  - **data.userNeedsToAcceptWorkshopLegalAgreement** (*boolean*) – Does the user need to accept the Steam Workshop legal agreement (**true**) or not (**false**)? See the Workshop Legal Agreement for more information.
- **err** (*boolean*): **true** if there was any IO error with the request.

Example:

```
Steam.UGC.createItem(Steam.utils.getAppID(), "Community", function(data, err)
    if err or data.result ~= 1 then
        print('Failure when creating item')
    else
        populateItem(data.publishedFileId)
    end
end)
```

UGC.**startItemUpdate**(*consumerAppId*, *publishedFileId*)

Parameters

- **consumerAppId** (*number*) – The App ID that will be using this item.
- **publishedFileId** (*uint64*) – The item to update.

Returns (*uint64*) A handle that you can use with future calls to modify the item before finally sending the update.

SteamWorks StartItemUpdate

Starts the item update process.

This gets you a handle that you can use to modify the item before finally sending off the update to the server with *UGC.submitItemUpdate()*.

Example:

```
local function populateItem(id)
    local handle = Steam.UGC.startItemUpdate(Steam.utils.getAppID(), id)
    Steam.UGC.setItemContent(handle, rootFolder)
    Steam.UGC.setItemTitle(handle, "My Item")
    Steam.UGC.setItemDescription(handle, "A Workshop item")
    Steam.UGC.setItemPreview(handle, rootFolder .. '/preview.png')
```

(continues on next page)

(continued from previous page)

```
    Steam.UGC.submitItemUpdate(handle, "First Revision", function(data, err)
        if err or data.result ~= 1 then
            print('Update failed')
        else
            print('Update successfull')
        end
    end)
end
```

UGC.**setItemContent**(*updateHandle*, *contentFolder*)

Parameters

- **updateHandle** (*uint64*) – The workshop item update handle to customize.
- **contentFolder** (*string*) – The absolute path to a local folder containing the content for the item.

Returns (*boolean*) **true** upon success. **false** if the UGC update handle is invalid.

SteamWorks SetItemContent

Sets the folder that will be stored as the content for an item.

For efficient upload and download, files should not be merged or compressed into single files (e.g. zip files).

Note: This must be set before you submit the UGC update handle using *UGC.submitItemUpdate()*.

Example:: See *UGC.startItemUpdate()*'s example.

UGC.**setItemDescription**(*updateHandle*, *description*)

Parameters

- **updateHandle** (*uint64*) – The workshop item update handle to customize.
- **description** (*string*) – The new description of the item.

Returns (*boolean*) **true** upon success. **false** if the UGC update handle is invalid.

SteamWorks SetItemDescription

Sets a new description for an item.

The description must be limited to the length defined by k_cchPublishedDocumentDescriptionMax.

You can set what language this is for by using `UGC.setItemUpdateLanguage()` **(missing)**, if no language is set then "english" is assumed.

Note: This must be set before you submit the UGC update handle using *UGC.submitItemUpdate()*.

Example:: See *UGC.startItemUpdate()*'s example.

UGC.**setItemPreview**(*updateHandle*, *previewFile*)

Parameters

- **updateHandle** (*uint64*) – The workshop item update handle to customize.
- **previewFile** (*string*) – The absolute path to a local preview image file for the item.

Returns **true** upon success. **false** if the UGC update handle is invalid.

SteamWorks SetItemPreview

Sets the primary preview image for the item.

The format should be one that both the web and the application (if necessary) can render. Suggested formats include JPG, PNG and GIF.

Note: This must be set before you submit the UGC update handle using *UGC.submitItemUpdate()*.

Example:: See *UGC.startItemUpdate()*'s example.

UGC.**setItemTitle**(*updateHandle*, *title*)

Parameters

- **updateHandle** (*uint64*) – The workshop item update handle to customize.
- **title** (*string*) – The new title of the item.

Returns (*boolean*) **true** upon success. **false** if the UGC update handle is invalid.

SteamWorks SetItemTitle

Sets a new title for an item.

The title must be limited to the size defined by k_cchPublishedDocumentTitleMax.

You can set what language this is for by using `UGC.setItemUpdateLanguage()`, if no language is set then "english" is assumed.

Note: This must be set before you submit the UGC update handle using *UGC.submitItemUpdate()*.

Example:: See *UGC.startItemUpdate()*'s example.

UGC.**submitItemUpdate**(*updateHandle*, *changeNote*, *callback*)

Parameters

- **updateHandle** (*uint64*) – The update handle to submit.
- **changeNote** (*string?*) – A brief description of the changes made (Optional, set to **nil** for no change note).
- **callback** (*function*) – Called asynchronously when this function returns. See below.

Returns nothing

SteamWorks SubmitItemUpdateResult

Uploads the changes made to an item to the Steam Workshop.

You can track the progress of an item update with *UGC.getItemUpdateProgress()*.

callback(data, err) receives two arguments:

- **data** (*table*) – Similar to SubmitItemUpdateResult_t, or **nil** if **err** is **true**.
  - **data.result** (*number*) – The result of the operation. See EResult.
  - **data.userNeedsToAcceptWorkshopLegalAgreement** (*boolean*) – Does the user need to accept the Steam Workshop legal agreement (**true**) or not (**false**)? See the Workshop Legal Agreement for more information.
- **err** (*boolean*): **true** if there was any IO error with the request.

Example:: See *UGC.startItemUpdate()*'s example.

UGC.**getNumSubscribedItems**()

Returns (*number*) Total number of subscribed items. **0** if called from a game server.

SteamWorks GetNumSubscribedItems

Gets the total number of items the current user is subscribed to for the game or application.

Example:

```
print('You are subscribed to ' .. Steam.UGC.getNumSubscribedItems() .. ' items')
```

UGC.**getSubscribedItems**()

Returns (*table*) An array of *PublishedFileId* (more precisely, *uint64*) for all your subscribed items. Empty if called from a game server.

SteamWorks GetSubscribedItems

Gets a list of all of the items the current user is subscribed to for the current game.

Warning: This function is slightly different from the SteamWorks API. You don't need to send the array, it is returned by the function.

Example:

```
for _, id in ipairs(Steam.UGC.getSubscribedItems()) do
    local flag = Steam.UGC.getItemState(id)
    if flag.installed then
        print('Subscribed item is installed!')
        local success, sizeOnDisk, folder = Steam.UGC.getItemInstallInfo(id)
        print('Install location: ' .. folder)
        print('Install size: ' .. sizeOnDisk)
    elseif flag.downloading then
        print('Subscribed item is downloading!')
    else
        print('Subscribed item is doing something')
    end
end
```

UGC.**getItemState**(*publishedFileId*)

Parameters **publishedFileId** (*uint64*) – The workshop item to get the state for.

Returns

(*table*) A table with flags for the item state, or nil if the item is not tracked on client. All flags are boolean values.

- **subscribed** – The current user is subscribed to this item. Not just cached.
- **legacyItem** – The item was created with the old workshop functions in ISteamRemoteStorage.
- **installed** – Item is installed and usable (but maybe out of date).
- **needsUpdate** – The items needs an update. Either because it's not installed yet or creator updated the content.
- **downloading** – The item update is currently downloading.

- **downloadPending** – `UGC.downloadItem()` **(missing)** was called for this item, the content isn't available until the callback is fired.

SteamWorks GetItemState

Gets the current state of a workshop item on this client.

Example:: See *`UGC.getSubscribedItems()`*'s example.

`UGC.`**`getItemInstallInfo`**(*id*)

Returns

(*boolean*) **true** if the operation is successfull. **false** in the following cases:

- cchFolderSize is 0.
- The workshop item has no content.
- The workshop item is not installed.

If this value is **false**, nothing else is returned. Otherwise:

Returns (*number*) Returns the size of the workshop item in bytes.

Returns (*string*) Returns the absolute path to the folder containing the content.

Returns (*number*) Returns the time when the workshop item was last updated.

SteamWorks GetItemInstallInfo

Gets info about currently installed content on the disc for workshop items that have `installed` set.

Calling this sets the "used" flag on the workshop item for the current player and adds it to their `usedOrPlayed` list.

If `legacyItem` is set then folder contains the path to the legacy file itself, not a folder.

Example:: See *`UGC.getSubscribedItems()`*'s example.

`UGC.`**`getItemUpdateProgress`**(*handle*)

Parameters **`handle`** (*`uint64`*) – The update handle to get the progress for.

Returns (*string*) The current status. One of 'Invalid', 'PreparingConfig', 'PreparingContent', 'UploadingContent', 'UploadingPreviewFile', 'CommittingChanges'. See EItemUpdateStatus.

Returns (*number*) The current number of bytes uploaded.

Returns (*number*) The total number of bytes that will be uploaded.

SteamWorks GetItemUpdateProgress

Gets the progress of an item update.

Example:

```
local rev = {
    PreparingConfig = 0,
    PreparingContent = 1,
    UploadingContent = 2,
    UploadingPreviewFile = 3,
    CommittingChanges = 4,
    Invalid = 5, -- also Invalid when the job is finished
}
local function get_progress(handle)
    local st, uploaded, total = Steam.UGC.getItemUpdateProgress(handle)
```

(continues on next page)

(continued from previous page)

```
    local p = rev[st] / 5
    -- total may be 0 depending on the status
    if total ~= 0 then
        p = p + 0.2 * (uploaded / total)
    end
    return p
end
```

UGC.**startPlaytimeTracking**(*vec*, *callback*)

Parameters

- **vec** (*table*) – The array of workshop items (*PublishedFileId*, more precisely *uint64*) you want to start tracking. (Maximum of 100 items.)
- **callback** (*function*) – Called asynchronously when this function returns. It is only called if you send between 1 and 100 items. See below.

Returns nothing

SteamWorks StartPlaytimeTracking

Start tracking playtime on a set of workshop items.

When your app shuts down, playtime tracking will automatically stop. **callback(data, err)** receives two arguments:

- **data** (*table*) – Similar to StartPlaytimeTrackingResult_t, or **nil** if **err** is **true**.
  - **data.result** (*number*) – The result of the operation. See EResult.
- **err** (*boolean*): **true** if there was any IO error with the request.

Example:

```
-- Tracks all subscribed items (you probably shouldn't do this)
Steam.UGC.startPlaytimeTracking(Steam.UGC.getSubscribedItems(), function(data, err)
    if not err and data.result == 1 then
        print('Tracking succeded')
    end
end)
```

UGC.**stopPlaytimeTracking**(*vec*, *callback*)

Parameters

- **vec** (*table*) – The array of workshop items (*PublishedFileId*, more precisely *uint64*) you want to stop tracking. (Maximum of 100 items.)
- **callback** (*function*) – Called asynchronously when this function returns. It is only called if you send between 1 and 100 items. See below.

Returns nothing

SteamWorks StopPlaytimeTracking

Stop tracking playtime on a set of workshop items.

When your app shuts down, playtime tracking will automatically stop.

callback(data, err) receives two arguments:

- **data** (*table*) – Similar to StopPlaytimeTrackingResult_t, or **nil** if **err** is **true**.

- **data.result** (*number*) – The result of the operation. See EResult.

- **err** (*boolean*): **true** if there was any IO error with the request.

Example:

```
local function stopTracking(...)
    Steam.UGC.stopPlaytimeTracking({...}, function(data, err)
        if not err and data.result == 1 then
            print('Tracking successfully stopped')
        end
    end)
end
```

UGC.**stopPlaytimeTrackingForAllItems**(*callback*)

Parameters **callback** (*function*) – Called asynchronously when this function returns. It must be of the same type as the callback in *UGC.stopPlaytimeTracking()*.

Returns nothing

SteamWorks StopPlaytimeTracking

Stop tracking playtime of all workshop items.

When your app shuts down, playtime tracking will automatically stop.

Example:

```
Steam.UGC.stopPlaytimeTrackingForAllItems(function(data, err)
    if not err and data.result == 1 then
        print('Tracking successfully stopped for all items')
    end
end)
```

CHAPTER 8

Extra

This module has some extra functions that are not in the SteamWorks API but may be useful to deal with it.

8.1 List of Functions

- *extra.parseUint64()*

8.2 Function Reference

extra.**parseUint64**(*str*)

Parameters **str** (*string*) – The string to convert to uint64.

Returns (*uint64*) The parsed number. 0 if the string was invalid.

See *64-bit integers*

Converts a string to a 64-bit integer that can be used with this library.

Example:

```
local function saveMyId()
    -- This is highly useless since your ID won't change, but you get the point.
    writeFile('my_id.txt', tostring(Steam.user.getSteamID()))
end

local function readMyId()
    return Steam.extra.parseUint64(readFile('my_id.txt'))
end
```

CHAPTER 9

Indices and tables

- genindex

Index