Steamworks Extension

This section is for those users that have been given access to the Steam API for publishing your game to that platform. To be able to use these functions you *must* have been accepted onto Steam previously, either through a publisher or through the self-publishing system.

Guides

This sections provides a variety of important guides to get you started using the extension:

- Setup Guide
- Migration Changes (what changed from the builtin Steam functionality)

Management

This extension provides the following management functions:

- steam_init (automatically called by the extension itself)
- steam_update REQUIRED
- steam_shutdown (automatically called by the extension itself)

Modules

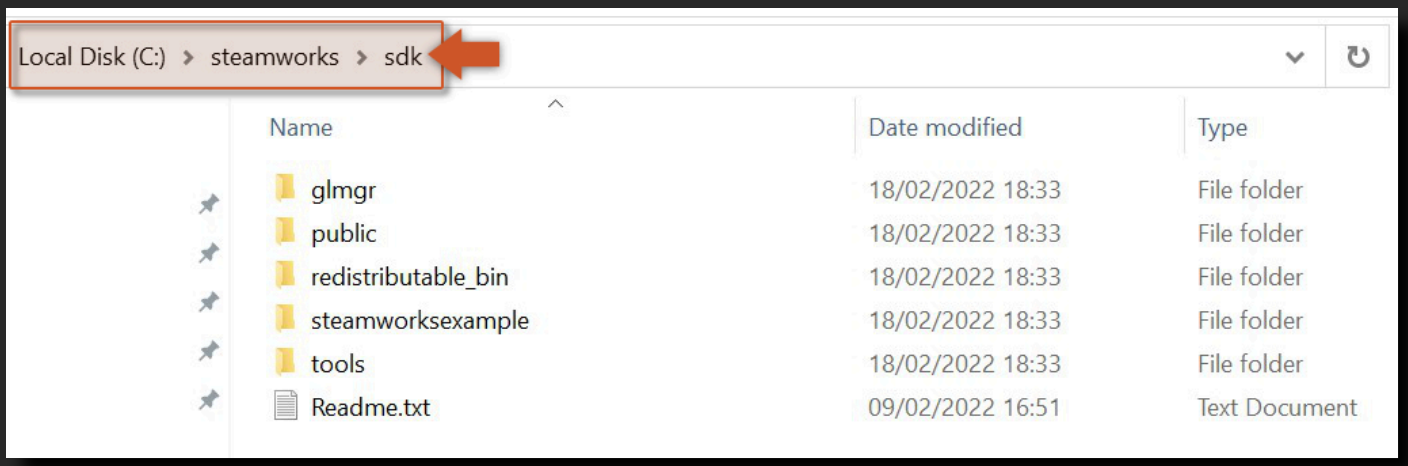
There are a great number of different functions related to the Steam API. We've split them up into the following sections to make it easier to navigate:

- General Steam API
- The Steam Overlay
- Leaderboards
- Achievements And Statistics
- Steam Cloud
- DLC (Downloadable Content)
- UGC (User Generated Content)

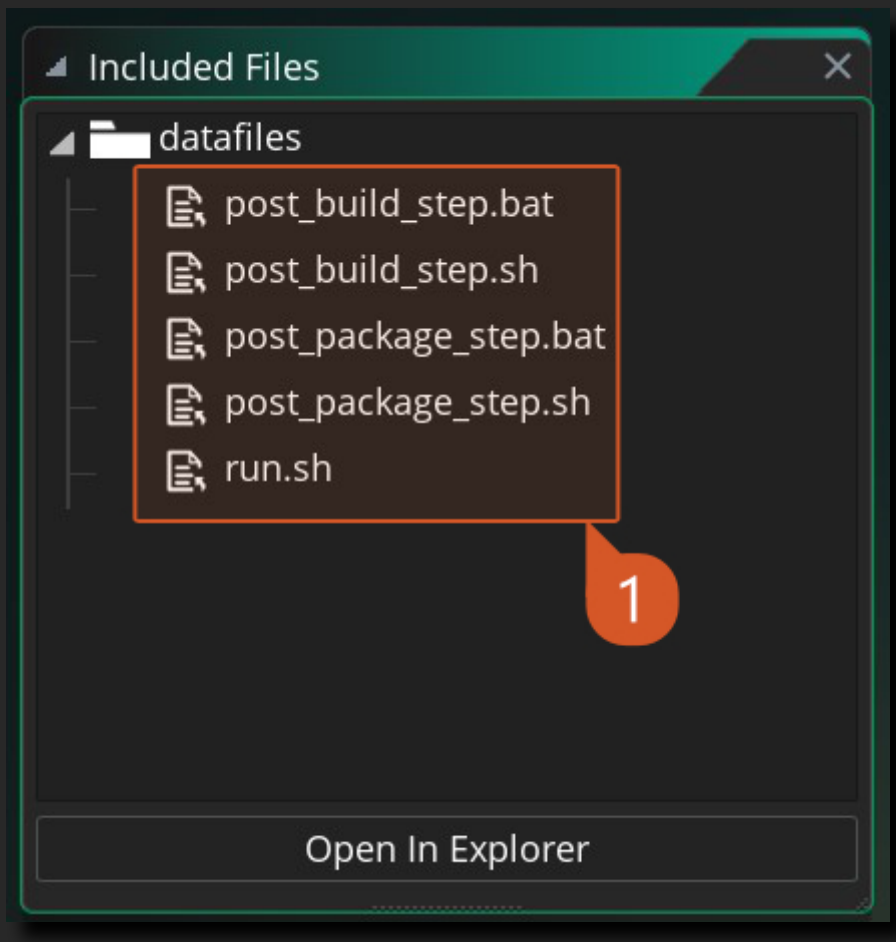
Setup Guide

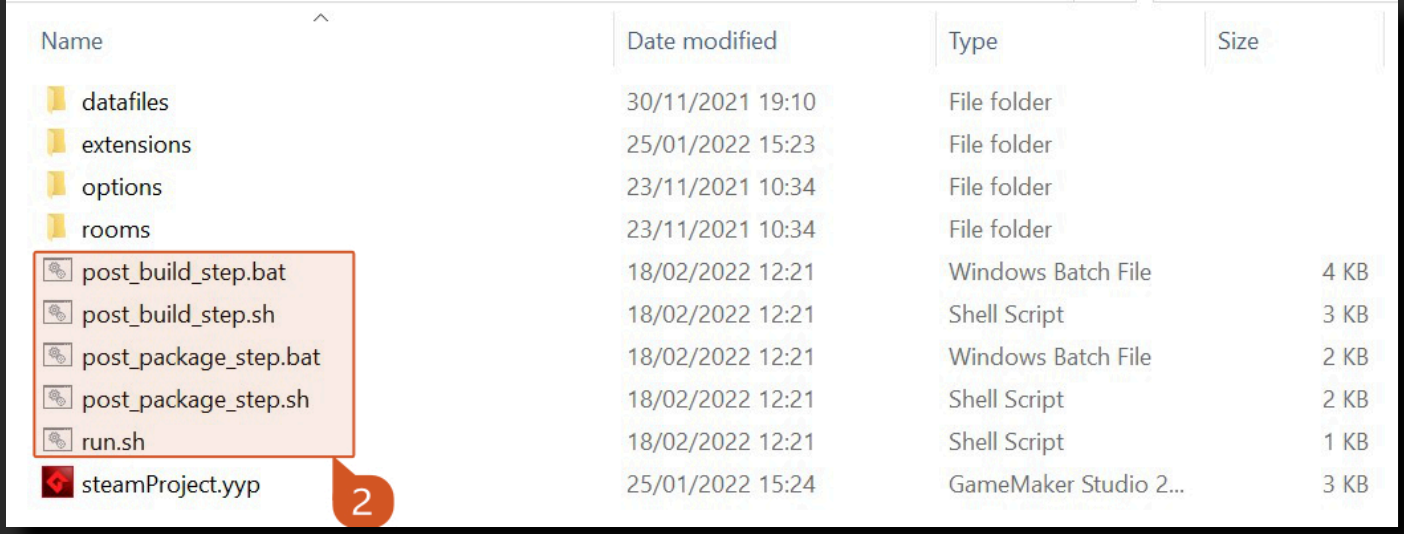
To use the Steam API extension you should follow these steps:

1. Steam app needs to be **installed**, **running** and with an account **logged in** (**official site**).
2. Download Steamworks SDK (1.53a) from Steam's **partner site** and extract the contents of the zip into a directory of your choice (e.g.: C:\steamworks\sdk).



3. Check your `<projectFolder>\datafiles` and move included files into your project's root folder next to your `.ypp` file.





NOTE Depending on the version of the extension (v1.0.10 and newer) `run.sh` file might not be present since it is not required anymore, if that is the case just ignore this file.

4. Go into your `<projectFolder>` and open `post_build_step.bat` (if you are on Windows) or `post_build_step.sh` (if you are on macOS) with a text editor and replace all the text after the `=` symbol with the path to the SDK installed in step 2.

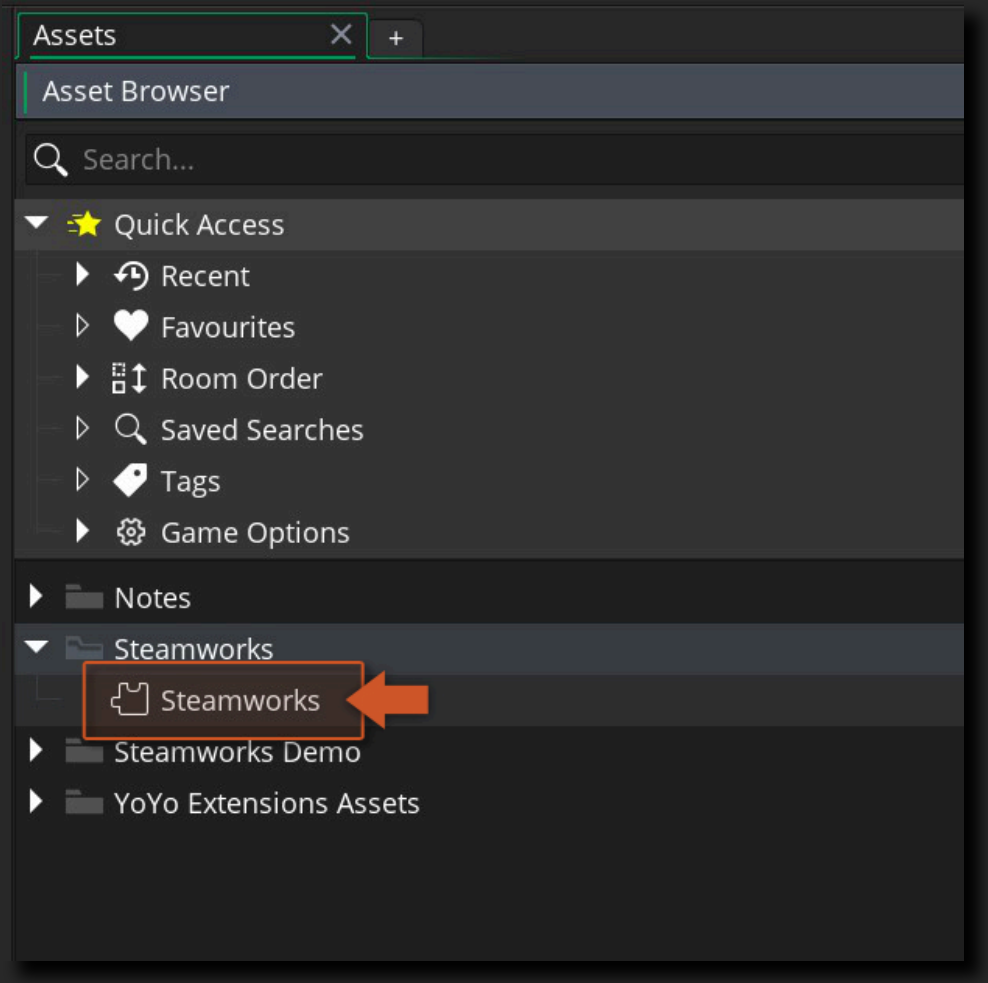
Windows

```
:: ########################################## EDIT VARIABLES ##########################################

:: Replace with your steamworks sdk path (download from here: https://partner.steamgames.com/doc/sdk)
set STEAM_SDK_PATH=C:\steamworks\sdk

:: ####################################################################################################
```

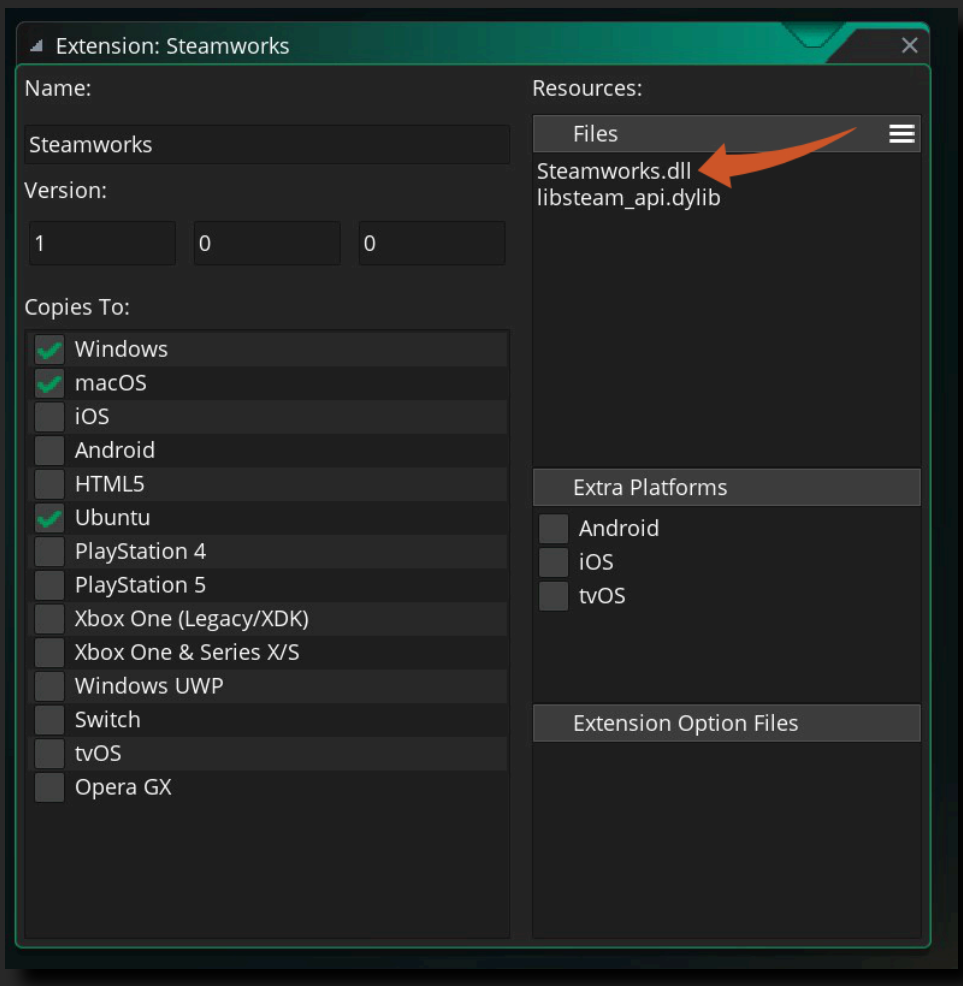
MacOS and Linux

```
# ########################################## EDIT VARIABLES ##########################################

# Replace with your steamworks sdk path (download from here: https://partner.steamgames.com/doc/sdk)
STEAM_SDK_PATH=/home/steamworks/sdk

# ####################################################################################################
```

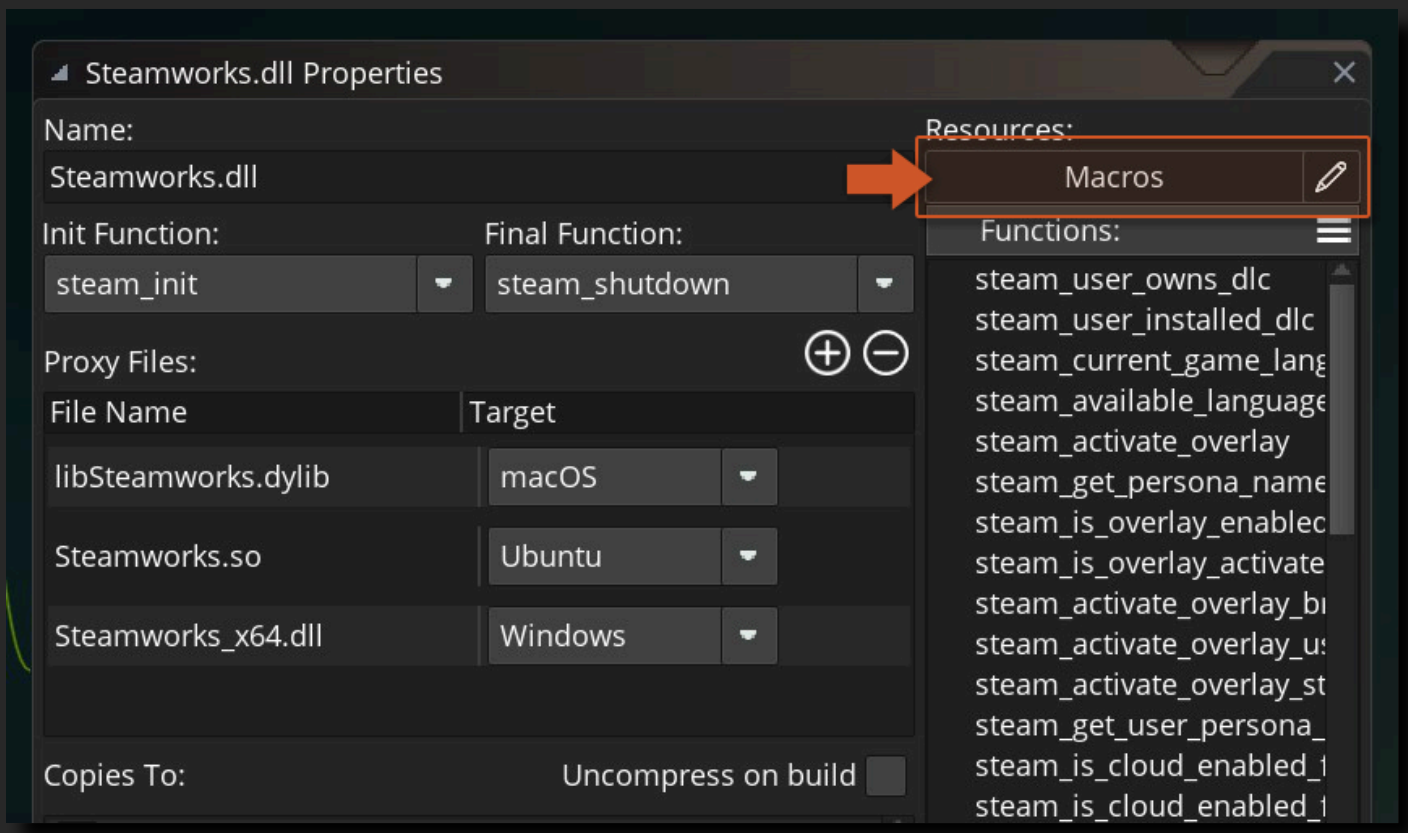
5. To setup your AppID and environment status, double click the Steamworks extension on your asset browser in the IDE.

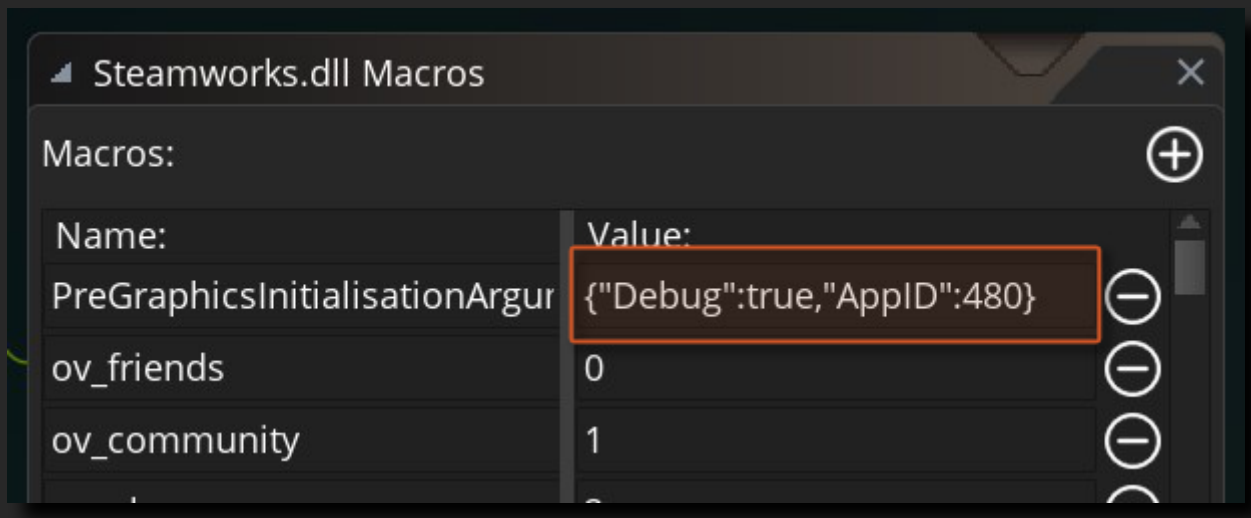


6. Here you can see the **Files** section. Double click on **Steamworks.dll**.



7. Now open the **Macros** window, and look for the `PreGraphicsInitialisationArgument` macro (it is defined as a struct). Here you can configure your **AppID** and set your development status with the **Debug** key.





8. You are now ready to use the extension in your Steam project.

NOTE If you set **Debug** key to `false` this will force your app to be launched by the Steam launcher. This should only be used when you are ready to send your app to production.

Migration Changes

During the migration of the Steamworks function library from the base GameMaker runner into this extension, there were some new functions that were added, and others that were slightly changed. This document covers the changes that happened during that migration.

Changed Functions

These are the functions that changed:

- steam_create_leaderboard

> This function is now asynchronous, meaning it will return an Async request ID that should be used inside a Steam Async Event to check when the task is finished.

New Functions

These are the new functions that were added to the Steam extension:

- steam_update REQUIRED
- steam_is_subscribed
- steam_set_warning_message_hook
- steam_upload_score_ext
- steam_upload_score_buffer_ext
- steam_ugc_delete_item

steam_init

This function initialises the steam APIs.

NOTE This function is already configured to be called at Game Start by the extension, and should not be called from your game code.

Syntax:

```
steam_init();
```

Returns:

```
N/A
```

steam_update

This function updates the steam APIs.

IMPORTANT This function is required to be called in order for the Steamworks extension to work. We recommend you place this function in a persistent controller object that calls it inside its **Step Event**.

Syntax:

```
steam_update();
```

Returns:

```
N/A
```

Example:

```
steam_update();
```

The above code will update the steam APIs.

steam_shutdown

This function shuts down the Steamworks API, releases pointers and frees memory.

NOTE This function is already configured to be called at Game End by the extension, and should not be called from your game code.

Syntax:

```
steam_shutdown();
```

Returns:

```
N/A
```

General

The following set of functions are all for checking the availability of certain aspects of the Steam client or server API. This means that these functions should be used before any other Steam API function call to ensure that the client/server setup is correct and communicating with your game:

- steam_initialised
- steam_stats_ready
- steam_get_app_id
- steam_get_user_account_id
- steam_get_user_steam_id
- steam_get_persona_name
- steam_get_user_persona_name
- steam_is_user_logged_on
- steam_current_game_language
- steam_available_languages
- steam_is_subscribed
- steam_set_warning_message_hook

steam_initialised

When using the Steam API, this function can be called to check that the Steam client API has been initialised correctly before any doing any further calls to Steam specific functions in your game.

Syntax:

```
steam_initialised();
```

Returns:

```
Bool
```

Example:

```
global.steam_api = false;
if (steam_initialised())
{
    if (steam_stats_ready() && steam_is_overlay_enabled())
    {
        global.steam_api = true;
    }
}
```

The above code will set a global variable to `true` if the Steam client API is correctly initialised, along with the Steam statistics and overlay functionality, or it will set the variable to `false` otherwise.

steam_stats_ready

When using the Steam API, this function can be called to check that the Steam client API has correctly initialised the statistics for your game.

Syntax:

```
steam_stats_ready();
```

Returns:

```
Bool
```

Example:

```
global.steam_api = false;
if steam_initialised()
{
    if steam_stats_ready() && steam_is_overlay_enabled()
    {
        global.steam_api = true;
    }
}
```

The above code will set a global variable to `true` if the Steam client API is correctly initialised, along with the Steam statistics and overlay functionality, or it will set the variable to `false` otherwise.

steam_get_app_id

This function is used retrieve the unique app ID that Steam assigns for your game, which is required for using some of the User Generated Content functions.

Syntax:

```
steam_get_app_id();
```

Returns:

```
Real
```

Example:

```
global.app_id = steam_get_app_id();
```

The above code gets the unique app ID for your game on Steam and stores it in a global variable.

steam_get_user_account_id

This function is used retrieve the unique User ID that Steam assigns to each user, which is required for using some of the User Generated Content functions.

Syntax:

```
steam_get_user_account_id();
```

Returns:

```
Real
```

Example:

```
global.user_id = steam_get_user_account_id();
```

The above code gets the unique user ID for the person who owns the game and stores it in a global variable.

steam_get_user_steam_id

You can use this function to return the unique Steam **user id** of the user currently logged into the Steam client. If you need to get the user's on screen user name you should refer to the function steam_get_persona_name.

Syntax:

```
steam_get_user_steam_id();
```

Returns:

```
int64
```

Example:

```
if steam_initialised()
{
    global.u_id = steam_get_user_steam_id();
}
```

The above code will set a global variable to the current users unique Steam ID if the Steam client API is correctly initialised.

steam_get_persona_name

You can use this function to return the user name of the user currently logged into the Steam client. This is the visible screen name and *not* the unique **user id** (this can be found using the function steam_get_user_steam_id).

Syntax:

```
steam_get_persona_name();
```

Returns:

```
String
```

Example:

```
if steam_initialised()
{
    global.p_name = steam_get_persona_name();
}
```

The above code will set a global variable to current users screen name if the Steam client API is correctly initialised.

steam_get_user_persona_name

This function can be used to retrieve the user name (screen name) for any specific user ID.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_get_user_persona_name(steamID);
```

Argument	Description
steamID	The unique Steam ID for a user.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"user_persona_name"`
steamid	int64	The unique user id of the user currently logged into the Steam client
persona_name	string	The visible screen name of the user currently logged into the Steam client

Example:

```
request = steam_get_user_persona_name(global.UGC_UserID);
```

The above code will request the user name of the user ID stored in the global variable "UGC_UserID", storing the returned value in a variable for parsing in the Async Event.

steam_is_user_logged_on

This function will return `true` if the Steam client currently has a live connection to the Steam servers. If it returns `false`, it means there is no active connection due to either a networking issue on the local machine, or the Steam server being down or busy.

Syntax:

```
steam_is_user_logged_on();
```

Returns:

```
Bool
```

Example:

```
if (steam_is_user_logged_on())
{
    global.user_id = steam_get_user_account_id();
}
```

The above code will check to see if the user is logged onto the Steam server and if it stores the user ID in a global variable.

steam_current_game_language

This function is used retrieve the current language that Steam is using (as a string), for example "english".

Syntax:

```
steam_current_game_language();
```

Returns:

```
String
```

Example:

```
language = steam_current_game_language();
```

The above code gets the language used for the current game.

steam_available_languages

This function can be used to retrieve a list of all available languages for Steam. The returned value is simply a a comma separated list of languages.

Syntax:

```
steam_available_languages();
```

Returns:

```
String
```

Example:

```
language = steam_available_languages();
```

The above gets the available languages for Steam as a string and stores it in a variable.

steam_is_subscribed

This function checks if the active user is subscribed to the current App ID.

> NOTE This will always return `true` if you're using Steam DRM.

Syntax:

```
steam_is_subscribed();
```

Returns:

```
Bool
```

Example:

```
if (steam_is_subscribed())
{
    show_debug_message("is_subscribed")
}
```

The above code will check to see if the user is logged onto the Steam server and if it stores the user ID in a global variable.

steam_set_warning_message_hook

This function sets a warning message hook to receive SteamAPI warnings and info messages in the console.

Syntax:

```
steam_set_warning_message_hook();
```

Returns:

```
N/A
```

Example:

```
steam_set_warning_message_hook();
```

The above code start Steamworks logging messages in console.

Overlay

The **Steam Overlay** is the visual display that can be brought up to display the Steam interface to the user. This is normally done by the user themselves using a combination of keys, but you also have the possibility of doing it from within your game, so that you can map a button or an event to show the overlay.

Functions

The extension provides you with the following functions:

- steam_is_overlay_enabled
- steam_is_overlay_activated
- steam_activate_overlay
- steam_activate_overlay_browser
- steam_activate_overlay_store
- steam_activate_overlay_user

Constants

This section also provides the following constants to use with the functions:

- OverlayType

steam_is_overlay_enabled

When using the Steam API, this function can be called to check that the Steam client API has the overlay functionality enabled.

Syntax:

```
steam_is_overlay_enabled();
```

Returns:

```
Bool
```

Example:

```
global.steam_api = false;
if steam_initialised()
{
    if steam_stats_ready() && steam_is_overlay_enabled()
    {
        global.steamapi = true;
    }
}
```

The above code will set a global variable to `true` if the Steam client API is correctly initialized, along with the Steam statistics and overlay functionality, or it will set the variable to `false` otherwise.

steam_is_overlay_activated

This function can be used to find out if the user has the Steam Overlay active or not. If the overlay is active and visible to the user the function will return `true`, and if it is not, then it will return `false`. An example of what this function can be used for would be for polling the Steam API for the overlay so that you can pause your game while the overlay is being shown.

Syntax:

```
steam_is_overlay_activated();
```

Returns:

```
Bool
```

Example:

```
if steam_is_overlay_activated()
{
    global.Pause = true;
}
```

The above code will check to see if the Steam overlay is active and if it is it will set the global variable "Pause" to true.

steam_activate_overlay

The Steam overlay is a piece of the Steam user interface that can be activated over the top of almost any game launched through Steam. It lets the user access their friends list, web browser, chat, and in-game DLC purchasing. The default key for a user to access the overlay while in a game is SHIFT + TAB, but you can also bring up any page of the overlay using this function. It takes one of six **constants** that are listed below:

Syntax:

```
steam_activate_overlay(overlay_type);
```

Argument	Type	Description
overlay_type	constant.OverlayType	The page index of the Steam API UI to show (see OverlayType constants).

Returns:

```
N/A
```

Example:

```
var key = keyboard_lastkey;
switch (key)
{
    case vk_f1: steam_activate_overlay(ov_friends); break;
    case vk_f2: steam_activate_overlay(ov_community); break;
    case vk_f3: steam_activate_overlay(ov_players); break;
    case vk_f4: steam_activate_overlay(ov_settings); break;
    case vk_f5: steam_activate_overlay(ov_gamegroup); break;
    case vk_f6: steam_activate_overlay(ov_achievements); break;
}
```

The above code polls the last keyboard key pressed and if it is any of the function keys from 1 to 6 it will open the corresponding page of the Steam overlay.

steam_activate_overlay_browser

With this function you can open the Steam game overlay to its web browser and then have it load the specified URL. you need to use the full URL as a string for this to resolve correctly, for example: `"http://www.steamgames.com/"`.

Syntax:

```
steam_activate_overlay(url);
```

Argument	Type	Description
url	string	The (full) URL for the overlay to open.

Returns:

```
N/A
```

Example:

```
if keyboard_check_pressed(vk_f1)
{
    steam_activate_overlay_browser("http://www.steamgames.com/");
}
```

The above code polls the keyboard for the F1 key and if it is then Steam overlay will be opened and resolve to the given URL.

steam_activate_overlay_store

With this function you can open the Steam overlay on the store page for a game so that users can buy or download DLC (for example). You need to supply the unique App ID for the game or DLC which you would get from the Steam dashboard when you set it up.

Syntax:

```
steam_activate_overlay_store(app_id);
```

Argument	Type	Description
app_id	integer	The unique App ID for your game.

Returns:

```
N/A
```

Example:

```
if keyboard_check_pressed(vk_f1)
{
    steam_activate_overlay_store(global.DLC_id);
}
```

The above code polls the keyboard for the F1 key and if it is then Steam overlay will be opened on the page for the game content using the app ID stored in the global variable.

steam_activate_overlay_user

This function will open the Steam overlay to one of the chosen dialogues relating to the user ID given.

Note that Steam IDs can be large numbers and so you may need to cast your ID value as an `int64()` before supplying it to the function.

Syntax:

```
steam_activate_overlay_user(dialog_name, steamid);
```

Argument	Type	Description
dialog_name	string	The dialogue to open the overlay on (see below).
steamid	int64	The Steam user ID or group ID to use.

Dialog Names	Description
`"steamid"`	Opens the Steam Community web browser to the page of the user or group
`"chat"`	Opens a chat window to the specified user

Returns:

```
N/A
```

Example:

```
var key = keyboard_lastkey;
switch (key)
{
    case vk_f1: steam_activate_overlay_user("steamid", global.GameGroupID); break;
```

```
    case vk_f2: steam_activate_overlay_user("chat", global.FriendID); break;
}
```

The above code polls the last keyboard key pressed and if it is function key 1 or function key 2, it will open the Steam overlay to either see the Steam group stored in the global variable "GamegroupID", or it will open the chat window to chat with the user stored in the global "FriendID" variable.

Overlay Type

These constants specify the type of overlay to be activated when using the function steam_activate_overlay.

Overlay Type Constant	Description
ov_friends	The friends page for the current user
ov_community	The community page for your game
ov_players	The page showing others that are playing the same game or that you have recently played with
ov_settings	The Steam client overlay settings
ov_gamegroup	Opens the Steam Community web browser to the official game group for this game
ov_achievements	The achievements page for your game

Leaderboards

The Steam API supports persistent leaderboards with automatically ordered entries. These leaderboards can be used to display global and friend leaderboards in your game and on the community web page for your game. Each game can have up to 10,000 leaderboards, and each leaderboard can be retrieved immediately after a player's score has been inserted into it, but note that for each leaderboard, a player can have only *one* entry, although there is no limit on the number of players per leaderboard.

Functions

Each leaderboard entry contains a name, a score and a rank for the leaderboard, and this data will be replaced when a new leaderboard entry is created for the user, and the following functions can be used to add and retrieve this data form the leaderboards for your game:

- steam_create_leaderboard
- steam_upload_score
- steam_upload_score_ext
- steam_upload_score_buffer
- steam_upload_score_buffer_ext
- steam_download_scores
- steam_download_scores_around_user
- steam_download_friends_scores

Data Types

The following data types are used by the leaderboard functions:

- LeaderboardEntry

Constants

The following constants are used by the leaderboard functions:

- LeaderboardDisplayType
- LeaderboardSortOrder

steam_create_leaderboard

With this function you can create a new leaderboard for your game. The first argument is a string which defines the name of your leaderboard, and this name should be used in any further function calls relating to the leaderboard being created. You can then define the sort order (see LeaderboardSortOrder constants) as well as the way in which the information is displayed (see LeaderboardDisplayType constants).

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If you have previously created a leaderboard with the same name (either through code or through your Steam page for the game) this function will not create a new one.

Syntax:

```
steam_create_leaderboard(lb_name, sort_oder, display_type);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are creating.
sort_oder	LeaderboardSortOrder constant	The method for sorting the leaderboard entries (see LeaderboardSortOrder constants).
display_type	LeaderboardDisplayType constant	The way to display the leaderboard to the user (see LeaderboardDisplayType constants).

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"create_leaderboard"`
status	real	The status code, `0` if the leaderboard was create and `1` if it already existed
lb_name	string	The name of the leaderboard

Example:

```
steam_create_leaderboard("Game Times", lb_sort_ascending, lb_disp_time_sec);
```

The above code will create a leaderboard called "Game Times", and set it to display the results in ascending order and with a display in seconds.

steam_upload_score

This function will send a score to the given leaderboard. The score to be uploaded is a real number, and the leaderboard name is a string that was defined when you created the leaderboard using the function steam_create_leaderboard.

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the Async event will not be triggered.

Syntax:

```
steam_upload_score(lb_name, score);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are uploading the scores to.
score	real	The score to upload.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
post_id	real	The asynchronous request ID
event_type	string	The string value `"leaderboard_upload"`

lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
success	bool	Whether or not the request was successful
updated	bool	Whether or not the leaderboard was updated (ie: the new score was better)
score	real	The score that was posted to the leaderboard

Extended Example:

In this example, we first upload a score and then parse the `async_load` map returned if successful. The code below shows a typical example for uploading:

```
if (hp <= 0)
{
    upload_ID = steam_upload_score("Game Scores", score);
    if (!upload_ID)
    {
        alarm[0] = room_speed;
    }
}
```

Note that we have set an alarm if the call fails. This would be used to try the upload again at a later time and you can add extra code there to retry the upload or to save the score to a text file should it continue to fail, etc... We now add the following into the **Steam Async Event** for the instance controlling the scores:

```
var type = ds_map_find_value(async_load, "event_type");
if (type == "leaderboard_upload")
{
    var lb_ID = ds_map_find_value(async_load, "post_id");
    if lb_ID == upload_ID
    {
        var lb_name = ds_map_find_value(async_load, "lb_name");
        var lb_done = ds_map_find_value(async_load, "success");
        var lb_score = ds_map_find_value(async_load, "score");
        var lb_updated = ds_map_find_value(async_load, "updated");
        show_debug_message("leaderboard post id:" + string(lb_ID) + " to lb:" +
string(lb_name) + " with score:" + string(lb_score) + " updated=" +
string(lb_updated));
        if (lb_done)
        {
            show_debug_message("- Succeeded");
        }
        else
        {
            show_debug_message("- Failed");
        }
```

```
    }
}
```

in the example we are simply outputting the return values to the compiler window as debug messages, but you can use this event to deal with the information in any way you choose.

steam_upload_score_ext

This function will send a score to the given leaderboard. It is similar to the function steam_upload_score but has an extra argument that will allow you to force the update of the score, as by default Steam only updates the score if it is better than the previous one.

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the Async event will not be triggered.

Syntax:

```
steam_upload_score(lb_name, score);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are uploading the scores to.
score	real	The score to upload.
forceUpdate	bool	Whether or not the value should be replaced.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	Type	Description

post_id	real	The asynchronous request ID
event_type	string	The string value `"leaderboard_upload"`
lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
success	bool	Whether or not the request was successful
updated	bool	Whether or not the leaderboard was updated (ie: the new score was better or `forceUpdate` was set to `true`)
score	real	The score that was posted to the leaderboard

Extended Example:

In this example, we first upload a score and then parse the `async_load` map returned if successful. The code below shows a typical example for uploading:

```
if (hp <= 0)
{
    upload_ID = steam_upload_score("Game Scores", score, true);
    if (!upload_ID)
    {
        alarm[0] = room_speed;
    }
}
```

Note that we have set an alarm if the call fails. This would be used to try the upload again at a later time and you can add extra code there to retry the upload or to save the score to a text file should it continue to fail, etc... We now add the following into the **Steam Async Event** for the instance controlling the scores:

```
var type = ds_map_find_value(async_load, "event_type");
if (type == "leaderboard_upload")
{
    var lb_ID = ds_map_find_value(async_load, "post_id");
    if lb_ID == upload_ID
    {
        var lb_name = ds_map_find_value(async_load, "lb_name");
        var lb_done = ds_map_find_value(async_load, "success");
        var lb_score = ds_map_find_value(async_load, "score");
        var lb_updated = ds_map_find_value(async_load, "updated");
        show_debug_message("leaderboard post id:" + string(lb_ID) + " to lb:" +
string(lb_name) + " with score:" + string(lb_score) + " updated=" +
string(lb_updated));
        if (lb_done)
        {
            show_debug_message("- Succeeded");
```

```
        }
        else
        {
            show_debug_message("- Failed");
        }
    }
}
```

in the example we are simply outputting the return values to the compiler window as debug messages, but you can use this event to deal with the information in any way you choose.

steam_upload_score_buffer

This function will send a score to the given leaderboard along with a data package created from a buffer. The buffer should be no more than 256 bytes in size - anything beyond that will be chopped off - and can contain any data you require. The score to be uploaded should be a real number, and the leaderboard name is a string that was defined when you created the leaderboard using the function steam_create_leaderboard.

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the Async event will not be triggered.

Syntax:

```
steam_upload_score_buffer(lb_name, score, buffer);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are uploading the scores to.
score	real	The score to upload.
buffer	id.buffer	The ID of the buffer to attach.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents

Key	Type	Description
post_id	real	The asynchronous request ID
event_type	string	The string value `"leaderboard_upload"`
lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
success	bool	Whether or not the request was successful
updated	bool	Whether or not the leaderboard was updated (ie: the new score was better). Note that if you score was not updated neither will be the data buffer.
score	real	The score that was posted to the leaderboard

Extended Example:

In this example, we first upload a score and then parse the `async_load` map returned if successful. The code below shows a typical example for uploading, with a buffer being created to hold a string telling us which level the score was uploaded from:

```
if (hp <= 0)
{
    var buff = buffer_create(256, buffer_fixed, 1 );
    buffer_write(buff, buffer_string, "Uploaded on level " + string(global.Level));
    upload_ID = steam_upload_score("Game Scores", score, buff);
    if (!upload_ID)
    {
        alarm[0] = room_speed;
    }
    buffer_delete(buff);
}
```

Note that we have set an alarm if the call fails. This would be used to try the upload again at a later time and you can add extra code there to retry the upload or to save the score to a text file should it continue to fail, etc... Also note that we immediately delete the buffer, since it is no longer required for the function. We now add the following into the **Steam Async Event** for the instance controlling the scores:

```
var type = ds_map_find_value(async_load, "event_type");
if (type == "leaderboard_upload")
{
    var lb_ID = ds_map_find_value(async_load, "post_id");
    if lb_ID == upload_ID
    {
```

```
        var lb_name = ds_map_find_value(async_load, "lb_name");
        var lb_done = ds_map_find_value(async_load, "success");
        var lb_score = ds_map_find_value(async_load, "score");
        var lb_updated = ds_map_find_value(async_load, "updated");
        show_debug_message("leaderboard post id:" + string(lb_ID) + " to lb:" +
string(lb_name) + " with score:" + string(lb_score) + " updated=" +
string(lb_updated));
        if (lb_done)
        {
            show_debug_message("- Succeeded");
        }
        else
        {
            show_debug_message("- Failed");
        }
    }
}
```

In the example we are simply outputting the return values to the compiler window as debug messages, but you can use this event to deal with the information in any way you choose.

steam_upload_score_buffer_ext

This function will send a score to the given leaderboard along with a data package created from a buffer. The buffer should be no more than 256 bytes in size - anything beyond that will be chopped off - and can contain any data you require. This function is similar to steam_upload_score_buffer but has an extra argument that will allow you to force the update of the score, as by default Steam only updates the score if it is better than the previous one.

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the Async event will not be triggered.

Syntax:

```
steam_upload_score_buffer(lb_name, score, buffer);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are uploading the scores to.
score	real	The score to upload.
buffer	id.buffer	The ID of the buffer to attach.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	Type	Description
post_id	real	The asynchronous request ID
event_type	string	The string value `"leaderboard_upload"`
lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
success	bool	Whether or not the request was successful
updated	bool	Whether or not the leaderboard was updated (ie: the new score was better or `forceUpdate` was set to `true`). Note that if you score was not updated neither will be the data buffer.
score	real	The score that was posted to the leaderboard

Extended Example:

In this example, we first upload a score and then parse the `async_load` map returned if successful. The code below shows a typical example for uploading, with a buffer being created to hold a string telling us which level the score was uploaded from:

```
if (hp <= 0)
{
    var buff = buffer_create(256, buffer_fixed, 1 );
    buffer_write(buff, buffer_string, "Uploaded on level " + string(global.Level));
    upload_ID = steam_upload_score_ext("Game Scores", score, buff, true);
    if (!upload_ID)
    {
        alarm[0] = room_speed;
    }
    buffer_delete(buff);
}
```

Note that we have set an alarm if the call fails. This would be used to try the upload again at a later time and you can add extra code there to retry the upload or to save the score to a text file should it continue to fail, etc... Also note that we immediately delete the buffer, since it is no longer required for the function. We now add the following into the **Steam Async Event** for the instance controlling the scores:

```
var type = ds_map_find_value(async_load, "event_type");
if (type == "leaderboard_upload")
{
    var lb_ID = ds_map_find_value(async_load, "post_id");
    if lb_ID == upload_ID
    {
        var lb_name = ds_map_find_value(async_load, "lb_name");
        var lb_done = ds_map_find_value(async_load, "success");
        var lb_score = ds_map_find_value(async_load, "score");
        var lb_updated = ds_map_find_value(async_load, "updated");
        show_debug_message("leaderboard post id:" + string(lb_ID) + " to lb:" +
string(lb_name) + " with score:" + string(lb_score) + " updated=" +
string(lb_updated));
        if (lb_done)
        {
            show_debug_message("- Succeeded");
        }
        else
        {
            show_debug_message("- Failed");
        }
    }
}
```

In the example we are simply outputting the return values to the compiler window as debug messages, but you can use this event to deal with the information in any way you choose.

steam_download_scores

This function is used retrieve a sequential range of leaderboard entries by leaderboard ranking. The `start_idx` and `end_idx` parameters control the requested range of ranks, for example, you can display the top 10 on a leaderboard for your game by setting the start value to 1 and the end value to 10. The leaderboard name is a string that was defined when you created the leaderboard using the function steam_create_leaderboard.

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the async event will not be triggered.

Syntax:

```
steam_download_scores(lb_name, start_idx, end_idx);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are downloading the scores from.
start_idx	integer	The start position within the leaderboard.
end_idx	integer	The end position within the leaderboard.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents

Key	Type	Description
id	real	The asynchronous request ID
event_type	string	The string value `"leaderboard_download"`
status	int64	The status code if download fails
lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
entries	string	A json formatted string with all the downloaded entries (see LeaderboardEntry for details)

Extended Example:

In this extended example we will request the top ten ranking for the given leaderboard and parse its results in the Steam Async Event. to start with we need to request the scores with the following code:

```
score_get = steam_download_scores("Game Scores", 1, 10);
```

This will send off a request to the Steam Server for the scores from the leaderboard "Game Scores", storing the **async id** of the request in the variable "score_get". this will then be handled in the **Steam Async Event** in the following way:

```
var async_id = ds_map_find_value(async_load, "id");
if async_id == score_get
{
    var entries = ds_map_find_value(async_load, "entries");
    var map = json_decode(entries);
    if ds_map_exists(map, "default")
    {
        ds_map_destroy(map);
        exit;
    }
    else
    {
        var list = ds_map_find_value(map, "entries");
        var len = ds_list_size(list);
        var entry;
        for(var i = 0; i < len; i++;)
        {
            entry = ds_list_find_value(list, i );
            steam_name[i] = ds_map_find_value(entry, "name");
            steam_score[i] = ds_map_find_value(entry, "score");
            steam_rank[i] = ds_map_find_value(entry, "rank");
            steam_data[i] = ds_map_find_value(entry, "data");
        }
    }
```

```
    ds_map_destroy(map)
}
```

What we do here is first check the "id" key of the special **async_load** DS map. If this value is the same as the value of the original call-back function (stored in the "score_get" variable) we then continue to process the data. The first thing we do is parse the `async_load` DS map for the key "entries" which will contain a JSON formatted string containing the leaderboard data. This JSON object is then decoded (see **json_decode**) as another **DS map**, and this new map id is stored in the variable "map".

This map is checked for the key "default" and if that is found then the map is destroyed and the event is exited. If no "default" key is found, the code will then parse the map to extract the necessary information about the leaderboard, by first extracting a DS list from the "entries" key of the DS map, and then looping through each entry of the list to get *another* DS map with the name, score and rank of each entry. These values are then stored to arrays.

Once the loop has finished, the JSON **DS map** is destroyed (which in turn destroys all the internal maps and lists). There is no need to destroy the **async_load** DS map as this is handled for you by GameMaker Studio 2.

steam_download_scores_around_user

This function is used to retrieve leaderboard entries relative the current users entry. The `range_start` parameter is the number of entries to retrieve *before* the current users entry, and the `range_end` parameter is the number of entries after the current user's entry, and the current user's entry is *always* included in the results. For example, if the current user is number 5 on a given leaderboard, then setting the start range to -2 and the end range to 2 will return 5 entries: 3 through 7. If there are not enough entries in the leaderboard before or after the user's entry, Steam will adjust the range start and end points trying to maintained the range size. For example, if the user is #1 on the leaderboard, start is set to -2, and end is set to 2, Steam will return the first 5 entries in the leaderboard.

This is an asynchronous function that will trigger the **Steam Async Event** when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the async event will not be triggered.

Syntax:

```
steam_download_scores_around_user(lb_name, range_start, range_end);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are downloading the scores from.
range_start	integer	The start position within the leaderboard.
range_end	integer	The end position within the leaderboard.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"leaderboard_download"`
status	int64	The status code if download fails
lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
entries	string	A json formatted string with all the downloaded entries (see LeaderboardEntry for details)

Example:

```
request_id = steam_download_scores_around_user("Game Scores", -4, 5);
```

This will send off a request to the Steam Server for a range of 10 scores from the leaderboard `"Game Scores"`, centered on the player and will store the **async id** of the request in the variable `request_id`. This will then be handled in the **Steam Async Event**, as shown in the Extended Example for steam_download_scores.

steam_download_friends_scores

With this function you can retrieve *only* the scores on the leaderboard that belong to those people that are marked as "friends" in the Steam client. So, if your leaderboard has 200 entries, and 50 of them are your friends, this function will retrieve only those 50 results. The leaderboard name is a string that was defined when you created the leaderboard using the function steam_create_leaderboard.

This is an asynchronous function that will trigger the Steam Async Event when the task is finished.

NOTE If the function call fails for any reason it will return -1 and the async event will not be triggered.

Syntax:

```
steam_download_friends_scores(lb_name);
```

Argument	Type	Description
lb_name	string	The name of the leaderboard that you are downloading the scores from.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID

event_type	string	The string value `"leaderboard_download"`
status	int64	The status code if download fails
lb_name	string	The name of the leaderboard
num_entries	real	The number of returned entries
entries	string	A json formatted string with all the downloaded entries (see LeaderboardEntry for details)

Example:

```
request_id = steam_download_friends_scores("Game Scores");
```

This will send off a request to the Steam Server for the users friends scores from the given leaderboard and will store the **async id** of the request in the variable `request_id`. This will then be handled in the Steam Async Event, as shown in the Extended Example for steam_download_scores.

Leaderboard Entry

A leaderboard entry is represented by a json formatted string that can be returned by the async callback event of the following functions:

- steam_download_scores
- steam_download_scores_around_user
- steam_download_friends_scores

This string can be decoded into a DS map (see json_decode, needs to be destroyed afterwards) or into a struct (see json_parse, recommended) and will provide the following members.

Key	Type	Description
rank	real	The rank of the entry on the specified leaderboard
data	string	The base64 encoded string with the data provided when uploading scores using the steam_upload_score_buffer or steam_upload_score_buffer_ex OPTIONAL
score	real	The score attributed to this entry
name	string	The display name of the player for this entry
userID	int64	The unique user id of the player for this entry

NOTE If steam_upload_score_buffer or steam_upload_score_buffer_ext were used to upload the score, the decoded entry will now have a `"data"` key so you can retrieve the data of the uploaded buffer (see the Steam Async Event extended code example for further details). This data will be base64 encoded and so you will need to use the function buffer_base64_decode on the data before reading from the buffer.

Leaderboard Display Type

These constants specify the display type of a leaderboard and should be used with the function steam_create_leaderboard.

Leaderboard Display Type Constant	Description
`lb_disp_none`	Show the leaderboard "as is".
`lb_disp_numeric`	Show the leaderboard as a numeric display.
`lb_disp_time_sec`	Show the leaderboard values as times, with the base value being seconds.
`lb_disp_time_ms`	Show the leaderboard values as times, with the base value being milliseconds

Leaderboard Sort Order

These constants specify the sort order of a leaderboard and should be used with the function steam_create_leaderboard.

Leaderboard Sort Order Constant	Description
`lb_sort_none`	No sorting. The information will be displayed "as is".
`lb_sort_ascending`	Sort the leaderboard in ascending order.
`lb_sort_descending`	Sort the leaderboard in descending order.

Stats and Achievements

The Steam Stats and Achievements API provides an easy way for your game to provide persistent, roaming achievement and statistics tracking for your users. The user's data is associated with their Steam account, and each user's achievements and statistics can be formatted and displayed in their Steam Community Profile.

> **NOTE** You must wait until `steam_stats_ready` has returned true, before attempting to read or write stats and achievements.

Achievements

In addition to providing highly-valued rewards to players of your games, achievements are useful for encouraging and rewarding teamwork and player interaction, providing extra dimensionality to the game objectives, and rewarding users for spending more of their time in-game, and as such it is recommended that your game has a few. They are easily set up from the Steam Dashboard, but will require that you create special Icons for them.

The following functions are provided for working with achievements:

- steam_set_achievement
- steam_get_achievement
- steam_clear_achievement

Statistics Functions

Statistics track fine-grained pieces of information, such as play time, number of power-ups used, etc. You may choose to use them simply for tracking internal game data - so that, for instance, you can grant an achievement based on multi-session game-play statistics collected from the user across multiple computers. Or, you can track interesting game data for display on the user's Steam Community page, where users can compare their own stats against their friends.

NOTE Previously to being used statistics must be initialized from the Steamworks control panel for your game.

The following functions are provided for working with statistics:

- steam_set_stat_int
- steam_set_stat_float
- steam_set_stat_avg_rate
- steam_get_stat_int
- steam_get_stat_float
- steam_get_stat_avg_rate

Debug Functions

The following functions are provided for debugging purposes and are not recommended in the production version of you game:

- steam_reset_all_stats
- steam_reset_all_stats_achievements

If the user is in **Offline Mode**, Steam keeps a local cache of the stats and achievement data so that the APIs can be use as normal. Any stats unable to be committed are saved for the next time the user is online. In the event that there have been modifications on more than one machine, Steam will automatically merge achievements and choose the set of stats that has had more progress. Because Steam keeps a local cache of stats data it is not necessary for the game to *also* keep a local cache of the data on disk, especially as such caches often come in conflict and when they do it looks to a users as if their progress has been reverted, which is a frustrating experience.

steam_set_achievement

With this function you can tell the Steam API to award ("set") an achievement for the player. These achievements should have been defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel. The Steam Game Overlay will display a notification panel to the user informing them of the achievement that they have received, unless the achievement has already been awarded, in which case nothing will happen.

Syntax:

```
steam_set_achievement(ach_name);
```

Argument	Type	Description
ach_name	string	The name of the achievement to set.

Returns:

```
N/A
```

Example:

```
if hp <= 0
{
    global.Deaths += 1;
    if global.Deaths == 10
    {
        if !steam_get_achievement("ach_Player_Dies_Ten_Times")
steam_set_achievement("ach_Player_Dies_Ten_Times");
    }
}
```

The above code will reward the player an achievement if the global variable "Deaths" is equal to 10 and if the achievement has not already been awarded.

steam_get_achievement

With this function you can check the Steam API to see if a specific achievement has been awarded. The achievement should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel.

Syntax:

```
steam_get_achievement(ach_name);
```

Argument	Type	Description
ach_name	string	The name of the achievement to get.

Returns:

```
Bool
```

Example:

```
if hp <= 0
{
    global.Deaths += 1;
    if global.Deaths == 10
    {
        if !steam_get_achievement("ach_Player_Dies_Ten_Times")
steam_set_achievement("ach_Player_Dies_Ten_Times");
    }
}
```

The above code will reward the player an achievement if the global variable "Deaths" is equal to 10 and if the achievement has not already been awarded.

steam_clear_achievement

With this function you can tell the Steam API to clear (reset) a specific achievement. The achievement should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel.

Syntax:

```
steam_clear_achievement(ach_name);
```

Argument	Type	Description
ach_name	string	The name of the achievement to clear.

Returns:

```
N/A
```

Example:

```
if mouse_check_button_pressed(mb_left)
{
    steam_clear_achievement("Ach_Game_Win");
    steam_clear_achievement("Ach_Died_10_Times");
    steam_clear_achievement("Ach_Killed_100_Enemies");
    steam_clear_achievement("Ach_Beat_Boss_Level_1");
}
```

The above code will reset the achievements of the game when the user clicks the left mouse button.

steam_set_stat_int

With this function you can set a specific statistic to a new, **signed integer**, value. The statistic should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel. Examples of when you could use this are for tracking how many times the player dies or for tracking progress towards an achievement.

Syntax:

```
steam_set_stat_int(stat_name, value);
```

Argument	Type	Description
stat_name	string	The name of the statistic to set.
value	integer	The value to set the stat to.

Returns:

```
N/A
```

Example:

```
xp += 100;
steam_set_stat_int("Total_XP", steam_get_stat_int("Total_XP") + 100);
if steam_get_stat_int("Total_XP") > 1000
{
    if !steam_get_achievement("Ach_1000XP") steam_set_achievement("Ach_1000XP");
}
```

The above code sets a statistic and then checks the final value for it to decide whether to award an achievement or not.

steam_set_stat_float

With this function you can set a specific statistic to a new, **floating point**, value. The statistic should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel. Examples of when you could use this are for tracking how far your player has travelled, or what percentage of the game is complete.

Syntax:

```
steam_set_stat_float(stat_name, value);
```

Argument	Type	Description
stat_name	string	The name of the statistic to set.
value	real	The value to set the stat to.

Returns:

```
N/A
```

Example:

```
var dist_pc = (dist / dist_max) * 100;
steam_set_stat_float("Travelled", dist_pc);
```

The above code calculates a percentage based on the distance travelled variable "dist" and the maximum distance you can travel "dist_max" and then sets the stat "Travelled" to the new value.

steam_set_stat_avg_rate

This function permits you to set an average statistic type with a "sliding window" effect on the average. The "session_count" value is the current value that you wish to average out, while the "session_length" is the amount of game time since the last call to the function. Please see the **extended Example** below for further details on how this can be used.

Syntax:

```
steam_set_stat_avg_rate(stat_name, session_count, session_length);
```

Argument	Type	Description
stat_name	string	The name of the statistic to set.
session_count	real	The value to get the average of.
session_length	real	The time that has been taken since the last time the stat was set.

Returns:

```
N/A
```

Extended Example:

Since the average stat function can be complex to understand, we will illustrate its use with the following example. Consider the case where you'd like to track an average statistic, such as "Points earned per hour". One approach would be to have two stats: an *integer* stat, "TotalPoints", and a *float* stat "TotalPlayTimeHours", and then divide the total points by the total time to get the "Points per Hour" value.

However, once the player has accumulated a significant amount of playtime, the calculated average will change extremely slowly, and the more the user plays the game, the less responsive that average will be. If the user has spent 100 hours playing the game, the calculated average will "lag" by about 50 hours of that, and if they increase their skill, they will not see the increase in "Points Per Hour" that they expect. To get around that we

can use a "sliding window" to only calculate the "Points per hour" for the last 10 hours played.

So, to use this function, we would need to create a Steam stat (in the control panel for the game on the Workshop) called "AvgPointsPerHour" and set its **Window** property to 10. Now in your game you would have to add some global variables into an instance at the start:

```
global.Points = 0;
global.Time = 0;
```

You would then have some controller object to count up the global "Time" variable in an alarm (for example) every second, while your game-play would affect the global "Points" variable. At regular intervals while playing (again, in a controller object, perhaps in an Alarm, or at intervals from polling the "Time" value) you would set the stat like this:

```
steam_set_stat_avg_rate("AvgPointsPerHour", global.Points, (global.Time / 3600));
global.Points = 0;
global.Time = 0;
```

Note that we divide time by 3600 since we want the time in *hours* and not in seconds, and afterward we reset the global "Points" variable and the global "Time" variable to 0 so that the next time the function is called, we get a new average for the statistic. Now, what Steam will do is take this value that you have sent and create an average value over the time that was set for our "window".

steam_get_stat_int

With this function you can get the value of a specific **signed integer** statistic. The statistic should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel.

Syntax:

```
steam_get_stat_int(stat_name);
```

Argument	Type	Description
stat_name	string	The name of the statistic to get.

Returns:

```
Real
```

Example:

```
xp += 100;
steam_set_stat_int("Total_XP", steam_get_stat_int("Total_XP") + 100);
if steam_get_stat_int("Total_XP") > 1000
{
    if !steam_get_achievement("Ach_1000XP") steam_set_achievement("Ach_1000XP");
}
```

The above code sets a statistic and then checks the final value for it to decide whether to award an achievement or not.

steam_get_stat_float

With this function you can get the value of a specific **floating point** statistic. The statistic should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel.

Syntax:

```
steam_get_stat_float(stat_name);
```

Argument	Type	Description
stat_name	string	The name of the statistic to get.

Returns:

```
Real
```

Example:

```
var dist_pc = (dist / dist_max) * 100;
if steam_get_stat_float("Travelled") < dist_pc
{
    steam_set_stat_int("Travelled", dist_pc);
}
```

The above code calculates a percentage based on the distance travelled variable "dist" and the maximum distance you can travel "dist_max". It then polls the current value for the statistic "Travelled" and if it is less than the calculated value, it sets the stat again.

steam_get_stat_avg_rate

With this function you can get the value of a specific **average** statistic. The statistic should have been previously defined on the Steamworks control panel accounts page for your game and the string that is passed to the function should match that used as the **API Name** on the control panel.

Syntax:

```
steam_get_stat_avg_rate(stat_name);
```

Argument	Type	Description
stat_name	string	The name of the statistic to get.

Returns:

```
Real
```

Example:

```
var avg = steam_get_stat_avg_rate("PointsPerHour");
draw_text(8, 8, "PPH = " + string(avg);
```

The above code gets the current value for the average statistic "PointsPerHour" and draws it on the screen.

steam_reset_all_stats

With this function you can reset all the statistics for the **current user** to their default values (as defined in the Steamworks control panel for your game). If need to also reset the achievement to their default values use the steam_reset_all_stats_achievements instead.

TIP It is recommended that you only use this function as a debug tool when developing your game.

Syntax:

```
steam_reset_all_stats();
```

Returns:

```
N/A
```

Example:

```
ini_open("Save.ini");
if global.Version != ini_read_real("Data", "Version", 0)
{
    ini_write_real("Data", "Version", global.Version);
    steam_reset_all_stats();
}
ini_close();
```

The above code checks a stored value in an ini file against that of a global variable and if they are different, it resets the statistics for the game.

steam_reset_all_stats_achievements

With this function you can reset all the statistics *and* achievements for the **current user** to their default values (as defined in the Steamworks control panel for your game). If you only need to reset the stats to their default values use the steam_reset_all_stats instead.

TIP It is recommended that you only use this function as a debug tool when developing your game.

Syntax:

```
steam_reset_all_stats_achievements();
```

Returns:

```
N/A
```

Example:

```
ini_open("Save.ini");
if global.Version != ini_read_real("Data", "Version", 0)
{
    ini_write_real("Data", "Version", global.Version);
    steam_reset_all_stats_achievements();
}
ini_close();
```

The above code checks a stored value in an ini file against that of a global variable and if they are different, it resets the statistics and achievements for the game.

Cloud

The Steam Cloud provides an easy and transparent remote file storage system for your game. All files written to disk using the cloud functions will be replicated to the Steam servers after the game exits. If the user then changes computers, the files will then be downloaded to the new computer before the game launches, meaning that the game can then access the files by reading them using the appropriate Steam functions. The Steam Client does the work of ensuring that the files are kept synchronized across all computers the user may be accessing.

> NOTE By default, the Cloud is **not** enabled for a game on Steamworks. it must be enabled previously from the 'Cloud' tab of the Steamworks game admin, where you should set the byte and file quota. The next time you publish your games Steamworks configuration, the Cloud storage will be ready to use.

The following functions can be used to access the Steam Cloud from within GameMaker Studio 2

- steam_is_cloud_enabled_for_app
- steam_is_cloud_enabled_for_account
- steam_get_quota_total
- steam_get_quota_free
- steam_file_exists
- steam_file_size
- steam_file_persisted
- steam_file_write
- steam_file_write_file
- steam_file_read
- steam_file_share
- steam_file_delete

steam_is_cloud_enabled_for_app

With this function you can check to make sure that the Steam Cloud service is enabled for your game. It will return `true` if it is and `false` otherwise.

IMPORTANT This does not automatically mean that you can use the Cloud functions as the user can switch off Cloud synchronization from their Steam Client. You can check this using the function `steam_is_cloud_enabled_for_account`, but, even if it is disabled for the user (and enabled for the game), the functions will still work to store and retrieve data from a local copy of all files, it will just not upload them to the cloud on the game end, nor synchronize on the game start.

Syntax:

```
steam_is_cloud_enabled_for_app();
```

Returns:

```
Bool
```

Example:

```
if (steam_is_cloud_enabled_for_app())
{
    quota = steam_get_quota_total();
}
```

The above code checks to see if the steam cloud is enabled for the game and if so it gets the size of the storage quota and stores it in a variable.

steam_is_cloud_enabled_for_account

With this function you can check to make sure that the Steam Cloud service is enabled by the user in their Steam Client settings. It will return `true` if it is and `false` otherwise.

IMPORTANT This does not automatically mean that you can store data to the Cloud, as it will also have to have been enabled for your game (you can check this using the function `steam_is_cloud_enabled_for_app`). If the Steam Cloud is enabled for your game, but the user has it switched off locally, you can still use the Cloud functions to store and retrieve data from a local copy of all files, it will just not upload them to the cloud on the game end, nor synchronize on the game start.

Syntax:

```
steam_is_cloud_enabled_for_account();
```

Returns:

```
Bool
```

Example:

```
if (steam_is_cloud_enabled_for_account())
{
    steam_file_share("Save.txt");
}
```

The above code checks to see if the user has the Steam Cloud enabled and if it returns `true`, it will then synchronize the given file.

steam_get_quota_total

When using the Steam Cloud to store and synchronize files, you must set up the *quota* of space that your game will need. This quota is enforced on each Cloud-enabled game, on a per-user-per-game basis, so, for example, if the quota for Game X is 1 megabyte, then each Steam account that owns Game X may store, at most, 1 megabyte of data associated with that game in the Cloud. Any other Cloud-enabled games that the user owns (say, Game Y) will not be affected by the data stored by Game X. The default quota for new Steamworks games is one gigabyte, but you can change this from the Steamworks control panel for your game.

NOTE Once the quota is exhausted file writes **will fail**. If you think it may be possible for the quota to be exhausted for the user of your game, you should create code to handle it, as by doing nothing you leave users in a situation where they are unable to fix things and that will lead to a poor experience of your game.

Syntax:

```
steam_get_quota_total();
```

Returns:

```
Real
```

Example:

```
if (steam_is_cloud_enabled_for_app())
{
    quota = steam_get_quota_total();
}
```

The above code checks to see if the steam cloud is enabled for the game and if so it gets the size of the storage quota and stores it in a variable.

steam_get_quota_free

With this function you can find out how much free space is left for the user of the Steam Cloud quota. The value returned is in *bytes*.

Syntax:

```
steam_get_quota_free();
```

Returns:

```
Real
```

Example:

```
if (steam_is_cloud_enabled_for_app())
{
    quota = steam_get_quota_free();
}
```

The above code checks to see if the steam cloud is enabled for the game and if so it gets the size of the free storage space and stores it in a variable.

steam_file_exists

With this function you can check to see if a file from the Steam Cloud exists or not, with a return value of `true` if it exists, or `false` otherwise.

Syntax:

```
steam_file_exists(filename);
```

Argument	Type	Description
filename	string	The name of the file to check for.

Returns:

```
Bool
```

Example:

```
if (steam_file_exists("Save.txt"))
{
    save_str = steam_file_read("Save.txt");
}
```

The above code checks to see if a file exists on the Steam Cloud and if it does, it opens it and reads its contents into the variable "save_str".

steam_file_size

With this function you can check the size of a file stored on the Steam Cloud. The returned real number is the size, in bytes, of the file.

Syntax:

```
steam_file_size(filename);
```

Argument	Type	Description
filename	string	The name of the file to check the size of.

Returns:

```
Real
```

Example:

```
file_bytes = steam_file_size("Save.txt");
```

The above code stores the size of a file from the Steam Cloud in the variable "file_bytes".

steam_file_persisted

With this function you can check the given file to see if it has been synchronized with the Steam Cloud. A return value of `true` means that it is, while `false` means it is not.

Syntax:

```
steam_file_persisted(filename);
```

Argument	Type	Description
filename	string	The name of the file to check.

Returns:

```
Bool
```

Example:

```
if (!steam_file_persisted("Save.txt"))
{
    steam_file_share("Save.txt");
}
```

The above code will check to see if a file has been stored to the Steam Cloud, and if it has not it will then synchronize it.

steam_file_write

You can use this function to write data to a file, which will then be synchronized with the Steam Cloud when the user exits the game. if the file does not exist, this function will create it for you, and if it does already exists, it will overwrite any data that is already stored within the file with the new data string. The function will return a value of 0 if it fails for whatever reason and a value greater than 0 if it succeeds.

Syntax:

```
steam_file_write(filename, data, size);
```

Argument	Type	Description
filename	string	The name of the file to write to.
data	string	The data to write (a string).
size	integer	the size of the data to be written.

Returns:

```
Real
```

Example:

```
var fname = "SaveData.txt";
var data = string(global.Level) + "|" + string(global.Points) + "|" +
string(global.HP);
var len = string_length(data);
steam_file_write_file(fname, data, len);
```

The above code will prepare a number of local variables and then use them to write to (or create) a file which will then be synchronized with the Steam Cloud.

steam_file_write_file

With this function you can copy the contents of a locally saved file to a file that is synchronized with the Steam Cloud. The local file *must exist* before using this function, and it will return a value of 0 if it fails for whatever reason and a value greater than 0 if it succeeds.

Syntax:

```
steam_file_write_file(steam_filename, local_filename);
```

Argument	Type	Description
steam_filename	string	The Steam Cloud file to copy over.
local_filename	string	The local file to use to copy from.

Returns:

```
real
```

Example:

```
steam_file_write_file("rm_koala.png", "Koala2.png");
```

The above code will copy the contents of the file "Koala2.png" to the Steam Cloud file "rm_koala.png".

steam_file_read

This function will read the contents of the given file into a string which can later be parsed in your game.

Syntax:

```
steam_file_read(filename);
```

Argument	Type	Description
filename	string	The name of the file to read from.

Returns:

```
String
```

Example:

```
if steam_file_exists("Save.txt")
{
    save_str = steam_file_read("Save.txt");
}
```

The above code checks to see if a file exists on the Steam Cloud and if it does, it opens it and reads its contents into the variable "save_str".

steam_file_share

With this function you can force your game to synchronize the given file with the Steam Cloud. This is not normally necessary due to the fact that the game will synchronize automatically at the end of the player's session, nor is it recommended by Steam, but it can be useful to ensure sensitive information is synchronized immediately. The function will return a value of 0 if it fails for whatever reason and a value greater than 0 if it succeeds.

Syntax:

```
steam_file_share(filename);
```

Argument	Type	Description
filename	string	The name of the file synchronize.

Returns:

```
Real
```

Example:

```
if (!steam_file_persisted("Save.txt"))
{
    steam_file_share("Save.txt");
}
```

The above code will check to see if a file has been stored to the Steam Cloud, and if it has not it will then synchronize it.

steam_file_delete

This function will delete the given file from the Steam Cloud. The function will return a value of 0 if it fails for whatever reason and a value greater than 0 if it succeeds.

Syntax:

```
steam_file_delete(filename);
```

Argument	Type	Description
filename	string	The name of the file delete.

Returns:

```
Real
```

Example:

```
if (steam_file_exists("Save.txt"))
{
    steam_file_delete("Save.txt");
}
```

The above code will check to see if a file exists, and if it does, it deletes the file from the Steam Cloud.

DLC

Steam supports both free and paid downloadable content (DLC), and in the Steam client, a game with downloadable content appears as a single application in the user's game list with the downloadable content viewable through the games properties dialog. Once owned, downloadable content is treated as an integral part of the game and Steam will automatically update the content when a patch is available and installs the content when the user installs the game.

Since this is all handled by the Steam servers and the configuration of any DLC is done through the Steamworks control panel, there are only a couple of functions necessary in GameMaker Studio 2 to check for this extra content:

- steam_user_owns_dlc
- steam_user_installed_dlc

steam_user_owns_dlc

If your game has DLC created for it, you can use this function to check whether the user has bought it before accessing any files associated with it. The function will return `true` (`1`) if the player owns the content, `false` (`0`) if they don't own it *or* the given DLC ID is invalid, or `-1` if they're not logged into Steam.

NOTE Even if the user owns the DLC it doesn't mean that they have it installed in their local account, so you should additionally use the function steam_user_installed_dlc to make sure that it is before using it.

Syntax:

```
steam_user_owns_dlc(dlc_id);
```

Argument	Type	Description
dlc_id	int64	The unique identifier for the DLC to be checked.

Returns:

```
Integer
```

Example:

```
global.Level_Max = 100;
if steam_user_owns_dlc(10354)
{
    if steam_user_installed_dlc(10354)
    {
        global.Level_max = 200;
    }
}
```

The above code will check to see if the user has bought, and installed, the DLC with the id 10354, and if so set a global variable to a different value.

steam_user_installed_dlc

If your game has DLC created for it, you can use this function to check and see whether the user has installed it before accessing any files associated with it. The function returns `true` if the player has the content installed, and `false` if the user does not, but note that the user must also own the DLC, so you should use the additional function of `steam_user_owns_dlc` to check that it is owned as well before using it.

Syntax:

```
steam_user_installed_dlc(dlc_id);
```

Argument	Type	Description
dlc_id	int64	The unique identifier for the DLC to be checked.

Returns:

```
Bool
```

Example:

```
global.Level_Max = 100;
if (steam_user_owns_dlc(10354))
{
    if (steam_user_installed_dlc(10354))
    {
        global.Level_max = 200;
    }
}
```

The above code will check to see if the user has bought, and installed, the DLC with the id 10354, and if so set a global variable to a different value.

UGC

This section is for those users that have been given access to the Steam API for publishing your game to that platform and that want to use the possibilities that the Steam Workshop and Community gives you for adding and generating user content in your projects. The simplest form of user generated content is the ability for the user to take and share screenshots, which is facilitated using the following two functions:

- steam_is_screenshot_requested
- steam_send_screenshot

Before using any of the built in functions for the Steam UGC (**U**ser **G**enerated **C**ontent) API you need to have set up your game correctly from the Steam dashboard and you should have read through the required documentation found here:

- Sharing User Generated Content

> NOTE You need to have your game accepted for the Steam online store and have access to the developer areas of the Steam API documentation.

All subscribed UGC items will be downloaded by the Steam client automatically, and you should have code in the Steam Asynchronous Event to catch this and store the ID of the UGC that has been downloaded for use in the other UGC functions.

> IMPORTANT Steam UGC IDs can be huge numbers This means that sometimes you may need to store these as a string rather than try and store them as a real value, especially if working with buffers or trying to write the value to a text file (since this will convert it to a simplified standard format like "6.6624e+003" which will cause issues being read back).

The normal workflow for getting UGC into your game would be as follows:

1. The user would subscribe to an item (either from your game using steam_ugc_subscribe_item or from the client/browser). If done from the game you are able to "listen" to the callback from the Steam Async Event.

2. When you get a successful subscription callback this means that your game is now downloading the UGC. You would then check if the item is installed (ie: download completed) with **steam_ugc_get_item_install_info**.
3. If the item is not completely installed, you can use **steam_ugc_get_item_update_info** to track the download progress.

The following sections explain all the functions required to get UGC functioning in GameMaker Studio 2:

Creating And Editing Content

The following functions are essentially "wrapper" functions for those supplied in the Steam API for creating and uploading content to their servers. As such, we recommend that you read over the linked Steam documentation before using them to gain a greater understanding of how they work: **Creating And Uploading Content**.

- **steam_ugc_create_item**
- **steam_ugc_delete_item**
- **steam_ugc_start_item_update**
- **steam_ugc_set_item_title**
- **steam_ugc_set_item_description**
- **steam_ugc_set_item_visibility**
- **steam_ugc_set_item_tags**
- **steam_ugc_set_item_content**
- **steam_ugc_set_item_preview**
- **steam_ugc_submit_item_update**
- **steam_ugc_get_item_update_progress**

Consuming Content

Once your user content has been created and the workshop has it available for download, people can subscribe to it through the Steam App or through the Web portal. However GameMaker Studio 2 also includes the following functions to use the Steam API for creating and canceling subscriptions as well as for getting information about what the user is subscribed to currently:

- steam_ugc_subscribe_item
- steam_ugc_unsubscribe_item
- steam_ugc_num_subscribed_items
- steam_ugc_get_subscribed_items
- steam_ugc_get_item_install_info
- steam_ugc_get_item_update_info
- steam_ugc_request_item_details

Querying Content

There are also a large number of functions available to query the Steam API about the UGC items available:

- steam_ugc_create_query_user
- steam_ugc_create_query_user_ex
- steam_ugc_create_query_all
- steam_ugc_create_query_all_ex
- steam_ugc_query_set_cloud_filename_filter
- steam_ugc_query_set_match_any_tag
- steam_ugc_query_set_search_text
- steam_ugc_query_set_ranked_by_trend_days

- steam_ugc_query_add_required_tag
- steam_ugc_query_add_excluded_tag
- steam_ugc_query_set_return_long_description
- steam_ugc_query_set_return_total_only
- steam_ugc_query_set_allow_cached_response
- steam_ugc_send_query

You can get a preview image of any UGC item from the workshop by using the function steam_ugc_send_query to get the preview file handle of the image, and then calling the following function:

- steam_ugc_download

Constants

This section also provides a set of constants to be used along side the functions provided above:

- UGCFileType
- UGCFileVisibility
- UGCListSortOrder
- UGCListType
- UGCMatchType
- UGCQueryType

steam_is_screenshot_requested

This function will poll the Steam API to see if the key for taking a screenshot of the game has been pressed. The function will only return `true` for one step (game tick) when the key is pressed, and will return `false` at all other times.

Please note that if the screenshot key is pressed, this function will only return `true` once for each step that it is pressed, and return `false` for any subsequent calls *within the same step*. For example, if a screenshot is requested in the current frame and you call this function in the Step event to find that out, you will get `true`; however, if you call it again in Draw GUI to check whether a screenshot was requested, you will get `false` as the function had already been "used up" in the Step event. To use the function's return value multiple times within the same frame, it is recommended to store it in a variable and read that instead of calling the function again.

NOTE This function does **not** take a screenshot for you. This only signals that the key has been pressed and you must use the GameMaker Studio 2 functions screen_save or screen_save_part to save a local copy of the file to be uploaded.

Syntax:

```
steam_is_screenshot_requested();
```

Returns:

```
Bool
```

Example:

```
if steam_is_screenshot_requested()
{
    var file = "Catch_The_Haggis_" + string(global.scrn_num) + ".png");
    screen_save(file)
    steam_send_screenshot(file, window_get_width(), window_get_height());
    global.scrn_num += 1;
}
```

The above code will poll the Steam API for a screenshot request and if it has been, a unique name for the image file will be generated, a screenshot will be taken, and the file will be sent to the Steam Community page for the user.

steam_send_screenshot

With this function you can upload a screenshot to the Steam Community profile page of the currently logged in user. The filename you supply is the name of the local file that was created when you took the screenshot using the GameMaker Studio 2 functions screen_save or screen_save_part. The width and height define the image size, and the function will return a value of 0 if it fails for whatever reason and a value greater than 0 if it succeeds.

Syntax:

```
steam_send_screenshot(filename, width, height);
```

Argument	Type	Description
filename	string	The name of the image file to upload.
width	real	The width of the image.
height	real	The height of the image.

Returns:

```
Real
```

Example:

```
if steam_is_screenshot_requested()
{
    var file = "Catch_The_Haggis_" + string(global.scrn_num) + ".png");
    screen_save(file)
    steam_send_screenshot(file, window_get_width(), window_get_height());
    global.scrn_num += 1;
}
```

The above code will poll the Steam API for a screenshot request and if it has been, a unique name for the image file will be generated, a screenshot will be taken, and the file will be sent to the Steam Community page for the user.

steam_ugc_create_item

This function is used to prepare the Workshop API and generate a published file ID for the item to be added. The function *must* be called before doing anything else with the item to be uploaded, as you will be required to use the unique published ID value that it returns in the Steam Async Event for updating.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_create_item(consumer_app_id, file_type);
```

Argument	Type	Description
consumer_app_id	integer	The unique App ID for your game on Steam.
file_type	constant.UGCFileType	One of the available file type constants (see UGCFileType constants).

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_create_item"`

result	real	This will either be the GML constant `ugc_result_success` or some other real number (see the Steam docs, for more details)
legal_agreement_required	bool	Will be `true` or `false` (see the Steam docs for more details)
published_file_id	int64	This key holds the unique published ID for the item (you may need to cast it using the `int64()` function)

Extended Example:

In this example we first call the function and store the async ID value in a variable:

```
var app_id = steam_get_app_id();
new_item = steam_ugc_create_item(app_id, ugc_filetype_community);
```

This would then send off a request to the Steam API to create the new Workshop item, generating an async event which we would deal with as follows:

```
var event_id = async_load[? "id"];
if event_id == new_item
{
    var type = async_load[? "event_type"];
    if type == "ugc_create_item"
    {
        global.Publish_ID = async_load[? "published_file_id"];
    }
}
```

The above code checks the event type and if it is "ugc_create_item" then it retrieves the published file ID and stores it in a global variable for future reference.

steam_ugc_delete_item

This function attempts to delete a previously published UGC item.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_delete_item(ugc_query_handle);
```

Argument	Type	Description
ugc_query_handle	real	The query handle to use.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_item_delete"`
result	real	This will either be the GML constant `ugc_result_success` or some other real number (see the Steam docs, for more details)

Example:

```
steam_ugc_delete_item(ugc_query_handle);
```

The above code creates a deletion request for `ugc_query_handle`.

steam_ugc_start_item_update

This function must be called before adding or updating information on a UGC item. You need to supply the unique App ID for your game on Steam, along with the unique published file ID that was returned for the item when you created it using the function steam_ugc_create_item. The function will return a unique update handle for the item, which you can then use in the UGC item functions to update (or add) information for uploading.

Syntax:

```
steam_ugc_start_item_update(consumer_app_id, published_file_id);
```

Argument	Type	Description
consumer_app_id	real	The unique App ID for your game on Steam.
published_file_id	int64	The unique published file ID value for the item.

Returns:

```
Real
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_set_item_title

This function will set the title to be used for the given item.

The function will return `true` if the API was successfully accessed and `false` if there was an issue.

Syntax:

```
steam_ugc_set_item_title(ugc_update_handle, title);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
title	string	The title (max 128 characters) to be used for the item.

Returns:

```
Bool
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_set_item_description

This function will set the description to be used for the given item.

The function will return `true` if the API was successfully accessed and `false` if there was an issue.

Syntax:

```
steam_ugc_set_item_description(ugc_update_handle, description);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
description	string	The description (max 8000 characters) to be used for the item.

Returns:

```
Bool
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_set_item_visibility

This function will set the visibility of the given item, using one of the UGCFileVisibility constants.

The function will return `true` if the API was successfully accessed and `false` if there was an issue.

Syntax:

```
steam_ugc_set_item_visibility(ugc_update_handle, visibility);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
visibility	constant.UGCFileVisibility	The visibility to be used for the item (see UGCFileVisibility constant)

Returns:

```
Bool
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_set_item_tags

This function will set the tags to be used for the given item. The tags should be added to a 1D array as string elements and the array passed to the function.

The function will return `true` if the API was successfully accessed and `false` if there was an issue.

Syntax:

```
steam_ugc_set_item_tags(ugc_update_handle, tags);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
tags	string	The tags (as an string json array) to be used for the item.

Returns:

```
Bool
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, string(tagArray));
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_set_item_content

This function will set the content path to be used for the given item, and it should be a relative path to the folder which contains the content files to upload - which in turn should be in the save are *or* the game bundle (ie: an included file).

The function will return `true` if the API was successfully accessed and `false` if there was an issue.

Syntax:

```
steam_ugc_set_item_content(ugc_update_handle, content);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
content	string	The content path to be used for the item

Returns:

```
Bool
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_set_item_preview

This function will set the preview image to be used for the given item. The image should be supplied as either a PNG, JPG or GIF format file with a maximum size of 1MB. The path to the image should be a relative path in the save are *or* the game bundle (ie: an included file).

The function will return `true` if the API was successfully accessed and `false` if there was an issue.

Syntax:

```
steam_ugc_set_item_preview(ugc_update_handle, preview);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
preview	string	The preview image (JPG, GIF or PNG - max size 1MB) to be used for the item.

Returns:

```
Bool
```

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
```

```
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_submit_item_update

This function will submit the UGC item indexed by the given handle to the Steam Workshop servers, adding the change notes to be used for the given item.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_submit_item_update(ugc_update_handle, change_note);
```

Argument	Type	Description
ugc_update_handle	real	The unique handle for the UGC to be updated (returned from steam_ugc_start_item_update)
change_note	string	The change notes to be used for the item.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_update_item"`
result	real	This will either be the GML constant `ugc_result_success` or some other real number (see

		the Steam docs, for more details)
legal_agreement_required	bool	Will be `true` or `false` (see the Steam docs for more details)

Example:

```
var app_id = steam_get_app_id();
var updateHandle = steam_ugc_start_item_update(app_id, global.Publish_ID);
steam_ugc_set_item_title(updateHandle, "My workshop item(3)!");
steam_ugc_set_item_description( updateHandle, "testing workshop...");
steam_ugc_set_item_visibility(updateHandle, ugc_visibility_public);
var tagArray;
tagArray[0] = "Test";
tagArray[1] = "New";
steam_ugc_set_item_tags(updateHandle, tagArray);
steam_ugc_set_item_preview(updateHandle, "promo.jpg");
steam_ugc_set_item_content(updateHandle, "WorkshopContent1");
requestId = steam_ugc_submit_item_update(updateHandle, "Version 1.2");
```

The above code gets the game ID, then uses that along with a previously stored published file ID to generate an update handle for the item. This handle is then used to update various pieces of information before the update is pushed to the Workshop servers.

steam_ugc_get_item_update_progress

This function can be used to track the update status for an item. You give the item handle (as returned by the function steam_ugc_start_item_update) and an empty DS map which will then be populated with the update information (see table below)

If there is an error the function will return `false` and the map will be empty, otherwise the function returns `true`.

Syntax:

```
steam_ugc_get_item_update_progress(ugc_update_handle, info_map);
```

Argument	Type	Description
ugc_update_handle	integer	The unique handle for the UGC to be updated.
info_map	id.dsmap	A (previously created) DS map index.

info_map Output Contents		
Key	**Type**	**Description**
status_code	real	The Steam status code
status_string	string	A string for the current status
bytes_processed	real	The bytes processed so far
bytes_total	real	The total number of bytes in the update

Returns:

```
Bool
```

Example:

```
var uploadMap = ds_map_create();
steam_ugc_get_item_update_progress(global.itemHandle, uploadMap);
var statusCode = uploadMap[? "status_code"];
var status = uploadMap[? "status_string"];
var processed = uploadMap[? "bytes_processed"];
var total = uploadMap[? "bytes_total"];
draw_text(32, 0, "Upload info for item:" + string(global.itemHandle));
draw_text(32, 15, "status code:" + string(statusCode));
draw_text(32, 30, "status:" + string(status));
draw_text(32, 45, "bytes processed:" +string(processed));
draw_text(32, 60, "bytes total:" + string( total));
ds_map_destroy(uploadMap);
```

The above code will query the upload status of the item indexed in the global variable "itemHandle", using a `DS Map` to store the information. This is then parsed and the resulting values drawn to the screen.

steam_ugc_subscribe_item

This function can be used to subscribe to a UGC item.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_subscribe_item(published_file_id);
```

Argument	Type	Description
published_file_id	int64	The unique file ID for the UGC to subscribe to.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_subscribe_item"`
result	real	This will either be the GML constant `ugc_result_success` or some other real number (see the Steam docs, for more details)
published_file_id	int64	This key holds the unique published ID for the item (you may need to cast it using the int64 function, before passing it to subsequent functions)

Example:

```
steam_sub = steam_ugc_subscribe_item(global.pubFileID);
```

The above code will subscribe (and download) the item with the file ID stored in the global variable "pubFileID".

steam_ugc_unsubscribe_item

This function can be used to unsubscribe from a UGC item.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_unsubscribe_item(published_file_id);
```

Argument	Type	Description
published_file_id	int64	The unique file ID for the UGC to unsubscribe from.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_unsubscribe_item"`
result	real	This will either be the GML constant `ugc_result_success` or some other real number (see the Steam docs, for more details)
published_file_id	int64	This key holds the unique published ID for the item (you may need to cast it using the int64 function)

Example:

```
steam_sub = steam_ugc_unsubscribe_item(global.pubFileID);
```

The above code will unsubscribe (and remove) the item with the file ID stored in the global variable "pubFileID".

steam_ugc_num_subscribed_items

This function can be used to get the number of items that the current user has subscribed to.

Syntax:

```
steam_ugc_num_subscribed_items();
```

Returns:

```
Real
```

Example:

```
numSub = steam_ugc_num_subscribed_items();
```

The above code will store the number of subscribed items in a variable.

steam_ugc_get_subscribed_items

This function will populate a DS list with all the published file IDs for the items that the user is currently subscribed to. You must first create the list and store the index in a variable, then pass this to the function.

The function will return `true` if everything is correct and the Steam API is initialized, or `false` if there is an error.

Syntax:

```
steam_ugc_get_subscribed_items(item_list);
```

Argument	Type	Description
item_list	id.dslist	A (previously created) DS list index.

Returns:

```
Bool
```

Example:

```
steam_list = ds_list_create();
steam_ugc_get_subscribed_items(steam_list);
```

The above code will create an empty DS list and then populate it with the file IDs for all subscribed items for the user.

steam_ugc_get_item_install_info

This function can be used to retrieve information about any given published file item that has been subscribed to and downloaded to the Steam local storage area for your game. You give the item ID and supply the index to an empty DS map which will then be populated with the install information (see table below).

If the item exists (ie.: as been subscribed and download was complete) then the function will return `true` and populate the map, otherwise it will return `false` and the map will remain empty.

Syntax:

```
steam_ugc_get_item_install_info(published_file_id, info_map);
```

Argument	Type	Description
published_file_id	int64	The unique handle for the UGC to be updated.
info_map	id.dsmap	A (previously created) DS map index.

info_map Output Contents		
Key	**Type**	**Description**
size_on_disk	real	The file size on disk (in bytes)
legacy_item	bool	Will be `true` or `false` depending on whether it is a legacy file or not
folder	string	This is the full path to the installed content (please refer to "Item Installation" in Steam SDK docs, as "legacy" items uploaded with the old method, are treated differently)

Returns:

```
Bool
```

Example:

```
var item_map = ds_map_create();
steam_ugc_get_item_install_info(global.fileID, item_map);
```

The above code will query the install status of the item indexed in the global variable "fileID", using a `DS Map` to store the information.

steam_ugc_get_item_update_info

This function can be used to retrieve information about the current download state for the given file ID. You give the item ID and supply the index to an empty DS map which will then be populated with the update information (see table below).

If the item exists then the function will return `true` and populate the map, otherwise it will return `false` and the map will remain empty.

Syntax:

```
steam_ugc_get_item_update_info(published_file_id, info_map);
```

Argument	Type	Description
published_file_id	int64	The unique file ID for the UGC to be checked.
info_map	id.dsmap	A (previously created) DS map index.

info_map Output Contents		
Key	**Type**	**Description**
needs_update	bool	Whether the item needs an update or not
is_downloading	bool	Whether the item is currently downloading or not
bytes_downloaded	real	The number of bytes that has been downloaded
bytes_total	real	The total size (number of bytes) required for the item on disk

Returns:

```
Bool
```

Example:

```
var info_map = ds_map_create();
var info = steam_ugc_get_item_update_info(global.fileID, info_map);
if info
{
    draw_text(32, 15, "needs_update: " + string(info_map[? "needs_update"]));
    draw_text(32, 30, "is_downloading: " + string(info_map[? "is_downloading"]));
    draw_text(32, 45, "bytes_downloaded: " + string(info_map[? "bytes_downloaded"]));
    draw_text(32, 60, "bytes_total: " + string(info_map[? "bytes_total"]));
}
```

The above code will query the download status of the item indexed in the global variable "fileID", using a `DS Map` to store the information.

steam_ugc_request_item_details

This function can be used to retrieve information about a given file ID. You give the file ID and supply a maximum age for checking (see the Steam docs for more information).

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_request_item_details(published_file_id, max_age_seconds);
```

Argument	Type	Description
published_file_id	real	The unique file ID for the UGC to be checked.
max_age_seconds	real	The age of the data to check (recommended 30 - 60 seconds).

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_item_details"`
result	real	This will either be the GML constant `ugc_result_success` or

		some other real number (see the Steam docs, for more details)
cached_data	bool	Will be `true` if the returned details are from the local cache or `false` if they are taken from the server
published_file_id	int64	This key holds the unique published ID for the item (you may need to cast it using the int64 function)
file_type	string	The type of file used
creator_app_id	real	The Steam ID of the item creator
consumer_app_id	real	The Steam ID of the item consumer
title	string	The title of the item
description	string	The description of the item
steam_id_owner	real	The Steam ID of the item owner
time_created	real	The time the item was first created
time_uploaded	real	The last time the item was updated
time_added_to_user_list	real	The time that the item was subscribed to
visibility	constant.UGCFileVisibility	The visibility of the item (see UGCFileVisibility constant)
banned	bool	Whether the item has been banned or not
accepted_for_use	bool	Whether the item has been accepted for use or not
tags_truncated	array	Short version of the tags as an array
tags	array	An array of the tags for the item
handle_file	int64	The unique file handle for the item
handle_preview_file	int64	The unique handle for the image preview for the item (can be used with

		steam_ugc_download to download a preview image)
filename	string	The name of the item file
file_size	real	The size of the item file
preview_file_size	real	The size of the preview image
url	string	The full URL for the item
up_votes	real	The number of up-votes received
down_votes	real	The number of down-votes received
score	real	The overall score of the item
account_id_owner	real	The account ID from the Steam ID owner (this can be used in function steam_ugc_create_query_user_ex)

Extended Example:

In this example we send off a details request for an item and then parse the resulting `async_load` DS map to set some variables. First we send of the request:

```
steam_details = steam_ugc_request_item_details(global.fileID, 60);
```

The above code will request details on the item with the file ID stored in the global variable and will trigger a Steam Async event with the returned information. In this event we can then parse the map and store some of the values in variables which can then be used to display the information to the user:

```
var map_id = async_load[? "id"];
var result = async_load[? "result"];
if (map_id == steam_details) && (result == ugc_result_success)
{
    mTitle = async_load[? "title"];
    mDesc = async_load[? "description"];
    mTags = async_load[? "tags"];
    m_hPreviewFile = async_load[? "handle_preview_file"];
    m_hOwnerSteamId = async_load[? "steam_id_owner"];
    mOwnerAccountId = async_load[? "account_id_owner"];
    mPubFileId = async_load[? "published_file_id"];
    mScore = async_load[? "score"];
}
```

steam_ugc_create_query_user

This function can be used to query the UGC data base. The function automatically uses the default ID for the app, user and assumes that the query is being done by the consumer (rather than the creator). The function requires you to use the following constants for the type of data to query (UGCListType), the type of item to match (UGCMatchType) and the order in which the returned items will be sorted (UGCListSortOrder), as well as a page number - note that a query will return a *maximum* number of 50 items.

The function returns a unique query handle value which should be stored in a variable for use in the other query functions. Note that this function only prepares the query but does not actually send it - for that you must call the function steam_ugc_send_query - and you can use further `steam_ugc_query_*()` functions to refine the search request before it is actually sent.

Syntax:

```
steam_ugc_create_query_user(list_type, match_type, sort_order, page);
```

Argument	Type	Description
list_type	constant.UGCListType	The type of data list to create (see UGCListType constants)
match_type	constant.UGCMatchType	The type of UGC items to query (see UGCMatchType constants)
sort_order	constant.UGCListSortOrder	The way that data should be ordered (see UGCListSortOrder constants)
page	real	The page number to query.

Returns:

```
Real
```

Example:

```
query_handle = steam_ugc_create_query_user(ugc_list_Published, ugc_match_Items,
ugc_sortorder_TitleAsc, 1);
```

The above code creates a query request and stores it's handle in a variable for future use.

steam_ugc_create_query_user_ex

This function can be used to query the UGC data base. The function requires the ID value for the user and the ID of the game that is going to consume the item and/or the ID of the game that created the item. You also need to use the following constants for the type of data to query (UGCListType), the type of item to query (UGCMatchType) and the order in which the returned items will be sorted (UGCListSortOrder), as well as a page number - note that a query will return a *maximum* number of 50 items.

The function returns a unique query handle value which should be stored in a variable for use in the other query functions. Note that this function only prepares the query but does not actually send it - for that you must call the function steam_ugc_send_query - and you can use further `steam_ugc_query_*()` functions to refine the search request before it is actually sent.

Syntax:

```
steam_ugc_create_query_user_ex(list_type, match_type, sort_order, page, account_id,
creator_app_id, consumer_app_id);
```

Argument	Type	Description
list_type	constant.UGCListType	The type of data list to create (see UGCListType constants)
match_type	constant.UGCMatchType	The type of UGC items to query (see UGCMatchType constants)
sort_order	constant.UGCListSortOrder	The way that data should be ordered (see UGCListSortOrder constants)
page	real	The page number to query
account_id	real	The Steam account ID
creator_app_id	real	The item creator app ID
consumer_app_id	real	The consumer app ID

Returns:

```
Real
```

Example:

```
query_handle = steam_ugc_create_query_user_ex(ugc_list_Published, ugc_match_Items,
ugc_sortorder_TitleAsc, page, global.AccountID, 0, global.GameID);
```

The above code creates a query request and stores it's handle in a variable for future use.

steam_ugc_create_query_all

This function can be used to query the UGC data base using some predefined query types. The function requires the following constants for the type of query to create (UGCQueryType), the type of item to match (UGCMatchType) and the page number to query - note that a query will return a *maximum* number of 50 items.

The function returns a unique query handle value which should be stored in a variable for use in the other query functions. Note that this function only prepares the query but does not actually send it - for that you must call the function steam_ugc_send_query - and you can use further `steam_ugc_query_*()` functions to refine the search request before it is actually sent.

Syntax:

```
steam_ugc_create_query_all(query_type, match_type, page);
```

Argument	Type	Description
query_type	constant.UGCQueryType	The type of query to create (see UGCQueryType constants)
match_type	constant.UGCMatchType	The type of UGC items to query (see UGCMatchType constants)
page	real	The page number to query

Returns:

```
Real
```

Example:

```
query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote, ugc_match_Items,
1);
```

The above code creates a general query request and stores it's handle in a variable for future use.

steam_ugc_create_query_all_ex

This function can be used to query the UGC data base. The function requires the ID of the game that is going to consume the item and/or the ID of the game that created the item, and you need to use the following constants for the type of query to create (UGCQueryType), the type of item to match (UGCMatchType) and the page number to query. Note that a query will return a *maximum* number of 50 items.

The function returns a unique query handle value which should be stored in a variable for use in the other query functions. Note that this function only prepares the query but does not actually send it - for that you must call the function steam_ugc_send_query - and you can use further `steam_ugc_query_*()` functions to refine the search request before it is actually sent.

Syntax:

```
steam_ugc_create_query_all_ex(query_type, match_type, page, creator_app_id,
consumer_app_id);
```

Argument	Type	Description
query_type	constant.UGCQueryType	The type of query to create (see UGCQueryType constants)
match_type	constant.UGCMatchType	The type of UGC items to query (see UGCMatchType constants)
page	real	The page number to query
creator_app_id	integer	The item creator app ID
consumer_app_id	integer	The consumer app ID

Returns:

```
Real
```

Example:

```
query_handle = steam_ugc_create_query_all_ex(ugc_query_RankedByVote, page,
global.AccountID, 0, global.GameID);
```

The above code creates a query request and stores it's handle in a variable for future use.

steam_ugc_query_set_cloud_filename_filter

This function can be used to further filter any given UGC query, specifically to check and see if a Workshop item file name must match or not. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is either `true` or `false` depending on whether you require the file names to match.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_cloud_filename_filter(ugc_query_handle, should_match);
```

Argument	Type	Description
ugc_query_handle	integer	The query handle to use.
match_cloud_filename	bool	Sets whether the UGC item file name should match or not.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote,
ugc_match_Items, 1);
steam_ugc_query_set_cloud_filename_filter(query_handle, true);
steam_ugc_query_add_excluded_tag(query_handle, "nasty chips");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_set_match_any_tag

This function can be used to further filter any given UGC query, specifically to switch on or off tag matching. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is either `true` or `false` depending on whether you require a check for matching tags.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_match_any_tag(ugc_query_handle, match_any_tag);
```

Argument	Type	Description
ugc_query_handle	integer	The query handle to use.
match_any_tag	bool	Sets whether the UGC item tags should match anything or not.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote,
ugc_match_Items, 1);
steam_ugc_query_set_match_any_tag(query_handle, false);
steam_ugc_query_add_excluded_tag(query_handle, "walking simulator");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_set_search_text

This function can be used to further filter any given UGC query, specifically to search for the given string in the title and description of the UGC items. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is a string you want to use as the search term.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_search_text(ugc_query_handle , search_text);
```

Argument	Type	Description
ugc_query_handle	real	The query handle to use.
search_text	string	The search text to use for the query.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote,
ugc_match_Items, 1);
steam_ugc_query_set_search_text(query_handle, "texture");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_set_ranked_by_trend_days

This function can be used to further filter any UGC query made using the `ugc_query_RankedByTrend` constant (UGCQueryType), specifically to search over a number of days. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is the number of days over which you want the query to run.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_ranked_by_trend_days(ugc_query_handle, days);
```

Argument	Type	Description
ugc_query_handle	real	The query handle to use.
days	real	The number of days to query.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByTrend,
ugc_match_Items, 1);
steam_ugc_query_set_ranked_by_trend_days(query_handle, 5);
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_add_required_tag

This function can be used to further filter any given UGC query, specifically to search only those UGC items with the given tag. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is a string you want to use as the tag to include.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_add_required_tag(ugc_query_handle, tag_name);
```

Argument	Type	Description
ugc_query_handle	integer	The query handle to use.
tag_name	string	The tag name to include.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByTrend,
ugc_match_Items, 1);
steam_ugc_query_add_required_tag(query_handle, "RPG");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_add_excluded_tag

This function can be used to further filter any given UGC query, specifically to exclude a given UGC from the query request. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is a string you want to use as the tag to exclude.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_add_excluded_tag(ugc_query_handle, tag_name);
```

Argument	Type	Description
ugc_query_handle	integer	The query handle to use.
tag_name	string	The tag name to exclude.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote,
ugc_match_Items, 1);
steam_ugc_query_add_excluded_tag(query_handle, "walking simulator");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_set_return_long_description

This function can be used to further filter any given UGC query, specifically to retrieve the long description text in the call back event triggered when the query was sent. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is either `true` or `false`.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_return_long_description(ugc_query_handle, should_return);
```

Argument	Type	Description
ugc_query_handle	real	The query handle to use.
long_description	bool	Whether to have the query return the long description text.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote,
ugc_match_Items, 1);
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_set_return_total_only

This function can be used to further filter any given UGC query, specifically to request only the number of results without any other information (meaning that the DS map generated by the send function will contain the key "num_results" without any further map data). The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is either `true` or `false`.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_return_total_only(ugc_query_handle , total_only);
```

Argument	Type	Description
ugc_query_handle	real	The query handle to use.
total_only	bool	Whether to have the query return only the total number of hits or not.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByVote,
ugc_match_Items, 1);
steam_ugc_query_set_return_total_only(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_query_set_allow_cached_response

This function can be used to further filter any given UGC query, specifically to request that the query check the local cache rather than online. The query handle is the value returned when you created the query (using, for example, steam_ugc_create_query_user) and the second argument is either `true` or `false`.

The function will return `true` if the query filter was correctly set, or `false` otherwise.

Syntax:

```
steam_ugc_query_set_allow_cached_response(ugc_query_handle , check_cache);
```

Argument	Type	Description
ugc_query_handle	integer	The query handle to use.
cache	bool	Whether to have the query check the local cache or not.

Returns:

```
Bool
```

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByTrend,
ugc_match_Items, 1);
steam_ugc_query_add_required_tag(query_handle, "RPG");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_send_query

This function can be used to send a query request. You first define the query using one of the following function:

- steam_ugc_create_query_all
- steam_ugc_create_query_all_ex
- steam_ugc_create_query_user
- steam_ugc_create_query_user_ex

which will return a query handle. This handle is then used to set filters etc.... before being used in this function to send off the query request.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_send_query(ugc_query_handle);
```

Argument	Type	Description
ugc_query_handle	real	The query handle to send.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	Type	Description

id	real	The asynchronous request ID
event_type	string	The string value `"ugc_query"`
result	real	This will either be the GML constant `ugc_result_success` or some other real number (see the Steam docs, for more details)
cached_data	bool	Will be `true` if the returned details are from the local cache or `false` if they are taken from the server
total_matching	real	The total number of matching results
num_results	real	The number of results returned (max 50)
results_list	id.dslist	A DS list index, where each list entry is a DS Map index containing details of the particular item (see table below)

item_info Contents		
Key	**Type**	**Description**
published_file_id	int64	This key holds the unique published ID for the item (you may need to cast it using the int64 function)
file_type	string	The type of file used
creator_app_id	real	The Steam ID of the item creator
consumer_app_id	real	The Steam ID of the item consumer
title	string	The title of the item
description	string	The description of the item
steam_id_owner	real	The Steam ID of the item owner
time_created	real	The time the item was first created
time_uploaded	real	The last time the item was updated

time_added_to_user_list	real	The time that the item was subscribed to
visibility	constant.UGCFileVisibility	The visibility of the item (see UGCFileVisibility constant)
banned	bool	Whether the item has been banned or not
accepted_for_use	bool	Whether the item has been accepted for use or not
tags_truncated	array	Short version of the tags as an array
tags	array	An array of the tags for the item
handle_file	int64	The unique file handle for the item
handle_preview_file	int64	The unique handle for the image preview for the item (can be used with steam_ugc_download to download a preview image)
filename	string	The name of the item file
file_size	real	The size of the item file
preview_file_size	real	The size of the preview image
url	string	The full URL for the item
up_votes	real	The number of up-votes received
down_votes	real	The number of down-votes received
score	real	The overall score of the item
account_id_owner	real	The account ID from the Steam ID owner (can be used in the function steam_ugc_create_query_user)

Example:

```
var query_handle = steam_ugc_create_query_all(ugc_query_RankedByTrend,
ugc_match_Items, 1);
steam_ugc_query_add_required_tag(query_handle, "RPG");
steam_ugc_query_set_return_long_description(query_handle, true);
steam_ugc_query_set_allow_cached_response(query_handle, true);
query_ID = steam_ugc_send_query(query_handle);
```

The above code creates a query request and stores it's handle in a local variable for future use in the rest of the functions which further define the query request before sending the query.

steam_ugc_download

With this function you can download a preview image for any given UGC item. The `ugc_handle` is the unique identifying value for the image (which you can get using the function steam_ugc_send_query), and the destination filename is the name (and local path within the Steam sandbox) that you wish to give the image file when the download is complete.

This is an asynchronous function that will return an asynchronous id and trigger the Steam Async Event when the task is finished.

Syntax:

```
steam_ugc_download(ugc_handle, dest_filename);
```

Argument	Type	Description
ugc_handle	int64	The unique handle for the preview to be downloaded.
dest_filename	string	The file name to save the preview with.

Returns:

```
Real
```

Triggers:

```
Asynchronous Steam Event
```

async_load Contents		
Key	**Type**	**Description**
id	real	The asynchronous request ID
event_type	string	The string value `"ugc_create_item"`
result	real	This will either be the GML constant `ugc_result_success` or some other real

		number (see the **Steam docs** under **EResult**, for more details)
original_filename	string	This key holds the original name of the image file *on the server* (a string)
dest_filename	string	This key holds the image file name you passed in (a string)
ugc_handle	integer	

Extended Example:

In this example we first call the function and store the async ID value in a variable:

```
steam_get = steam_ugc_download(steam_handle, "\UGC\Preview_file.png");
```

This would then send off a file request to the Steam API, generating an async event which we would deal with as follows:

```
var event_id = async_load[? "id"];
if event_id == steam_get
{
    var type = async_load[? "event_type"];
    if type == "ugc_download"
    {
        sprite_delete(preview_sprite);
        preview_sprite = sprite_add(async_load[? "dest_filename"], 0, false, false,
0, 0);
    }
}
```

The above code checks the event type and then creates a sprite from the downloaded image.

UGC File Type

These constants specify the way that a shared file will be shared with the community and should be used while creating a new item with steam_ugc_create_item.

NOTE See Steam Docs for more details.

UGC File Type Constant	Description
`ugc_filetype_community`	This is used to create files that will be uploaded and made available to anyone in the community
`ugc_filetype_microtrans`	This is used to describe files that are uploaded but intended only for the game to consider adding as official content

UGC File Visibility

These constants specify possible visibility states that a Workshop item can be in. They are used with the function steam_ugc_set_item_visibility and are an async callback parameter for the functions steam_ugc_request_item_details and steam_ugc_send_query.

NOTE See Steam Docs for more details.

UGC File Visibility Constant	Description
ugc_visibility_public	Set the item to be publicly visible
ugc_visibility_friends_only	Set the item to be visible to only people on the users friends list
ugc_visibility_private	Set the item to be private

UGC List Sort Order

These constants specify the sorting order of user published UGC lists from queries created using one of the following functions:

- steam_ugc_create_query_user
- steam_ugc_create_query_user_ex

NOTE See Steam UGC Docs for more details.

UGC List Sort Order Constant	Description
`ugc_sortorder_CreationOrderDesc`	Returns items by creation date. Descending - the newest items are first
`ugc_sortorder_CreationOrderAsc`	Returns items by creation date. Ascending - the oldest items are first
`ugc_sortorder_TitleAsc`	Returns items by name
`ugc_sortorder_LastUpdatedDesc`	Returns the most recently updated items first
`ugc_sortorder_SubscriptionDateDesc`	Returns the most recently subscribed items first
`ugc_sortorder_VoteScoreDesc`	Returns the items with the more recent score updates first
`ugc_sortorder_ForModeration`	Returns the items that have been reported for moderation

UGC List Type

These constants specify the type of published UGC list to obtain from queries created using one of the following functions:

- steam_ugc_create_query_user
- steam_ugc_create_query_user_ex

NOTE See Steam UGC Docs for more details.

UGC List Type Constant	Description
ugc_list_Published	List of files the user has published
ugc_list_VotedOn	List of files the user has voted on. Includes both VotedUp and VotedDown
ugc_list_VotedUp	List of files the user has voted up (restricted to the current user only)
ugc_list_VotedDown	List of files the user has voted down (restricted to the current user only)
ugc_list_WillVoteLater	DEPRECATED
ugc_list_Favorited	List of files the user has favorited
ugc_list_Subscribed	List of files the user has subscribed to (restricted to the current user only)
ugc_list_UsedOrPlayed	List of files the user has spent time in game with
ugc_list_Followed	List of files the user is following updates for

UGC Match Type

These constants specify the types of UGC to obtain from queries created using one of the following function:

- steam_ugc_create_query_all
- steam_ugc_create_query_all_ex
- steam_ugc_create_query_user
- steam_ugc_create_query_user_ex

NOTE See Steam UGC Docs for more details.

UGC Match Type Constant	Description
`ugc_match_Items`	Both microtransaction items and Ready-to-use items
`ugc_match_Items_Mtx`	Microtransaction items
`ugc_match_Items_ReadyToUse`	Regular in game items that players have uploaded
`ugc_match_Collections`	Shared collections of UGC
`ugc_match_Artwork`	Artwork which has been shared
`ugc_match_Videos`	Videos which have been shared
`ugc_match_Screenshots`	Screenshots which have been shared
`ugc_match_AllGuides`	Both web guides and integrated guides
`ugc_match_WebGuides`	Guides that are only available on the steam community
`ugc_match_IntegratedGuides`	Guides that you can use within your game (like Dota 2's in game character guides)
`ugc_match_UsableInGame`	Ready-to-use items and integrated guides
`ugc_match_ControllerBindings`	Controller Bindings which have been shared

UGC Query Type (Sorting & Filtering)

These constants specify the sorting and filtering for queries across all available UGC, and are to be used with the following functions:

- steam_ugc_create_query_all
- steam_ugc_create_query_all_ex

NOTE See Steam UGC Docs for more details.

UGC Query Type Constant	Description
ugc_query_RankedByVote	Sort by vote popularity all-time
ugc_query_RankedByPublicationDate	Sort by publication date descending
ugc_query_AcceptedForGameRankedByAcceptanceDate	Sort by date accepted (for mtx items)
ugc_query_RankedByTrend	Sort by vote popularity within the given "trend" period (set in steam_ugc_query_set_ranked_by_tr
ugc_query_FavoritedByFriendsRankedByPublicationDate	Filter to items the user's friends have favorited, sorted by publication date descending
ugc_query_CreatedByFriendsRankedByPublicationDate	Filter to items created by friends, sorte publication date descending
ugc_query_RankedByNumTimesReported	Sort by report weight descending
ugc_query_CreatedByFollowedUsersRankedByPublicationDate	Filter to items created by users that th current user has followed, sorted by publication date descending
ugc_query_NotYetRated	Filtered to the user's voting queue
ugc_query_RankedByTotalVotesAsc	Sort by total # of votes ascending (use internally for building the user's voting queue)
ugc_query_RankedByVotesUp	Sort by number of votes up descendin use the "trend" period if specified (set in steam_ugc_query_set_ranked_by_tr
ugc_query_RankedByTextSearch	Sort by keyword text search relevancy