GXC Library

Welcome to the GXC Library! This is a marketplace asset containing functions for interacting with the GXC API. You can use this to submit challenge scores and retrieve information about the user's profile.

This manual only contains information on using the functions given in this library. You can view the **Opera GX section** on the YoYo Games Helpdesk for general guides on using GXC and creating challenges on GXC DevCloud.

Modules

Please look at the following sections for information on the different modules present in this library:

- General
- Challenge
- Profile

Miscellaneous

Please also read the following pages containing crucial information on the GXC functions:

- Callbacks & Async Events
- Response Error Codes
- Library Errors

General Functions

This asset contains the following general functions:

- gxc_get_query_param

gxc_get_query_param

When your game is played on GXC, some extra parameters are passed into the URL of the game so they can be retrieved in-game. This function will return the value of the parameter specified in the first argument as a string.

The following parameter keys can be specified in the `key` argument:

Key	Value
game	The ID of the game
track	The ID of the track that is being played
challenge	The ID of the currently active challenge, or `undefined` if no challenge is active
username	The username of the current user
userId	The ID of the current user
avatarUrl	A URL to the current user's avatar image (can be used with `sprite_add()` to be displayed in-game)

Note that if the specified parameter is not present in the URL, this function will return `undefined`.

Syntax:

```
gxc_get_query_param(key);
```

Argument	Description
key	The parameter key to get the value of

Returns:

```
String (or undefined)
```

Example:

```
var _current_challenge = gxc_get_query_param("challenge");
var _highscore_challenge = "34esa3a1-e41e-4a9f-aaaa-4da7bd24ada2";

if (_current_challenge == _highscore_challenge)
{
    gxc_submit_challenge_score(global.highscore);
}
```

The above code retrieves the ID of the currently active challenge and checks whether it's equal to a specific challenge ID (meaning that challenge is active). In that case, it submits the current highscore as a new score to the challenge.

Here is an example where the profile data is retrieved and drawn in-game:

```
// Create event
username = gxc_get_query_param("username");

var _avatar_url = gxc_get_query_param("avatarUrl");

user_sprite = sprite_add(_avatar_url, 0, 0, 0, 0, 0);

// Draw event
if (sprite_exists(user_sprite))
{
    draw_sprite(user_sprite, 0, x, y);
    draw_text(x + 100, y, username);
}
```

The "Create event" code above retrieves the username and avatar URL for the current user, then uses `sprite_add()` to download the avatar image through the given URL and stores its new ID in a variable. In the Draw event, it checks whether the user sprite exists (which will be `true` after it has been downloaded) and if it does, draws the sprite along with the username.

Challenges

Overview

A GXC game can have several challenges, only one of which can be active when the game is being played. The functions given below are used to interact with the currently active challenge (if there is one).

Note that the currently active challenge can be returned by calling `gxc_get_query_param("challenge")`.

Functions

The following functions are given for working with challenges:

- gxc_submit_challenge_score

gxc_submit_challenge_score

Overview

This function is used to submit a new score to the currently active challenge. You specify the score value to submit to the challenge, and an optional callback method which is called when a response arrives from the server. As explained on the Callbacks page, if you don't specify this argument, an Async Social event will be triggered when a response is received.

The function will always return a request ID, which can be used to identify the request in an Async Social event if a callback method is not specified.

Callback Arguments

The callback method will receive two arguments: `_status` and `_result`, where the former is the status code for the HTTP response and the latter is a struct containing all returned data. This `_result` struct (or the `async_load` map in the Async Social event if a callback is not specified) will contain the following keys:

Key	Type	Value
data	array	An array containing the top 5 scores of the current user for the active challenge
errors	array	An array containing the errors that occurred for this request

The `data` array contains all the scores submitted by the user for the current challenge (up to 5), sorted from best-to-worst (so the best score is the first in the array). Each score is its own struct and contains the following keys:

Key	Type	Value
achievementDate	string	The date and time of the score being submitted
countryCode	string	The country code of the user
score	real	The score value
scoreId	string	The ID of the submitted score
userId	string	The ID of the user
username	string	The user's name

For example, you would access the achievement date of the best score by writing `_result.data[0].achievementDate`. You can also use `array_length()` to get the total number of items in the `data` array and loop through them.

The `errors` array contains structs indicating errors with the request. Each error struct will have a `code` variable containing an error message, all of which are listed on this page.

Syntax:

```
gxc_submit_challenge_score(score, [callback]);
```

Argument	Description
score	The new score value for the currently active challenge; should be an integer, if a decimal value is provided it will be rounded
callback	(Optional) A callback method which is called when an HTTP response is received

Returns:

```
Real (HTTP Request ID)
```

Example:

```
var _highscore_challenge = "34esa3a1-e41e-4a9f-aaaa-4da7bd24ada2";

if (gxc_get_query_param("challenge") == _highscore_challenge)
{
    gxc_submit_challenge_score(global.highscore);
}
```

The above code stores the ID of a challenge (retrieved from the GXC DevCloud website) and checks whether that challenge is currently active (by retrieving the `"challenge"` parameter from the URL). If that challenge is active, it submits the current highscore to the challenge, without a callback method (which is not required if all you need to do is submit a score).

If you need to confirm that the score was uploaded and then perform a certain action, you can make use of the callback function:

```
if (gxc_get_query_param("challenge") == _highscore_challenge)
{
    instance_create_layer(0, 0, "GUI", obj_loading_bar);

    gxc_submit_challenge_score(global.highscore, function (_status, _result)
    {
        if (_status == 200)
        {
            instance_destroy(obj_loading_bar);
            instance_create_layer(0, 0, "GUI", obj_upload_success);
        }
    });
}
```

The above code creates a "loading bar" instance before submitting the score, and uses a callback function to check whether the score was successfully uploaded to the server. In that case, it destroys the loading bar and creates an "upload success" object. (Note that ideally you would also want to handle what happens if an error is returned.)

The status code is being checked against `200` as that indicates that the request was successful.

Profile Functions

The following functions are given for working with profiles:

- gxc_get_profile

gxc_get_profile

Overview

This function is used to retrieve information about the current user's profile. You can specify an optional callback method which is called when a response arrives from the server. As explained on the Callbacks page, if you don't specify this argument, an Async Social event will be triggered when a response is received.

> *NOTE: The data retrieved using this function is more readily available in the URL parameters, which can be retrieved using `gxc_get_query_param()`. (Note that URL parameters only work when the game is being played on GXC and not locally.)*

The function will always return a request ID, which can be used to identify the request in an Async Social event if a callback method is not specified.

Callback Arguments

The callback method will receive two arguments: `_status` and `_result`, where the former is the status code for the HTTP response and the latter is a struct containing all returned data. This `_result` struct (or the `async_load` map in the Async Social event if a callback is not specified) will contain the following keys:

Key	Type	Value
data	struct	A struct containing information about the current user
errors	array	An array containing the errors that occurred for this request

The `data` struct will contain the following keys:

Key	Type	Value
username	string	The username of the current user
userId	string	The ID of the current user
avatarUrl	string	A URL to the current user's avatar image (can be used with `sprite_add()` to be displayed in-game)

The `errors` array contains structs indicating errors with the request. Each error struct will have a `code` variable containing an error message, all of which are listed on this page.

Syntax:

```
gxc_get_profile([callback]);
```

Argument	Description
callback	(Optional) A callback method which is called when an HTTP response is received

Returns:

```
Real (HTTP Request ID)
```

Example:

```
gxc_get_profile( function(_status, _result)
{
    if (_status == 200)
    {
        username = _result.data.username;

        var _avatar_url = _result.data.avatarUrl;

        user_sprite = sprite_add(_avatar_url, 0, 0, 0, 0, 0);
    }
});
```

The above code sends a request for the user's profile data, and in the callback function retrieves the username and the avatar URL. It then uses `sprite_add()` to download the avatar image through the given URL and stores its new ID in a variable.

Note that the status code is being checked against `200` as that indicates that the request was successful.

Callbacks & Async Events

Several functions in this library accept an optional callback method as an argument. If it is specified, it will get two arguments: `_status` and `_result`, where `_status` is the status of the HTTP response and `_result` is a struct containing all information returned from the response.

```
gxc_get_profile( function(_status, _result)
{
    if (_status == 200) show_debug_message("The user's name is " +
_result.data.username);
});
```

However, you also have the option to use the Async Social event instead of a callback function. To do so, you would simply not specify a callback argument in the function (or specify `undefined`) and then listen for a response in an Async Social event instead. (Note that if you do specify a valid callback method, then the Async Social event will **not** be fired.)

```
// Create event
profile_request = gxc_get_profile();

// Async - Social event
if (async_load[? "id"] == profile_request && async_load[? "success"])
{
    var _data = async_load[? "data"]
    show_debug_message("The user's name is " + _data.username);
}
```

It is recommended to use a callback method as the Async event approach is only provided for ease of use with Drag And Drop™.

Response Error Codes

Some callback methods may receive an `errors` array in their "result" struct (or the `async_load` map in case of an Async event), which will contain one or more error structs indicating any errors with the request. The table below lists all possible error strings that can be included in the `code` variable of such an error struct:

Error String	Description
"game_not_found"	The given game ID was invalid
"challenge_not_found"	The given challenge ID was invalid
"track_not_found"	The given track ID was invalid
"challenge_not_active"	The challenge has not been activated
"invalid_hash"	The given hash is incorrect

Library Errors

Functions in this library may sometimes throw errors if they come across something problematic, all of which are described below:

Error	Description
`required query params not found`	A function was trying to retrieve a query parameter (such as `"game"` or `"challenge"`) which was not found in the game's current URL
`param 'callback' must be of type method`	A value of the wrong type was passed into a function's `callback` argument, which only accepts methods; please look at the function's reference page for an example