GameJolt Extension

By implementing the Game Jolt Game API you can add trophies, leaderboards, cloud data storage, and sessions to your games to get players coming back for more!

See the official page for more documentation

Setup

Follow these guides to get yourself going on everything you need for your new game.

- GameJolt Setup
- Project Setup

Modules

This Game API presents a variety of modules that can be used to push your game to the next level. These are the included modules:

- Users (use for user authentication, REQUIRED)
- DataStorage
- Friends
- Scores
- Sessions
- Time
- Trophies

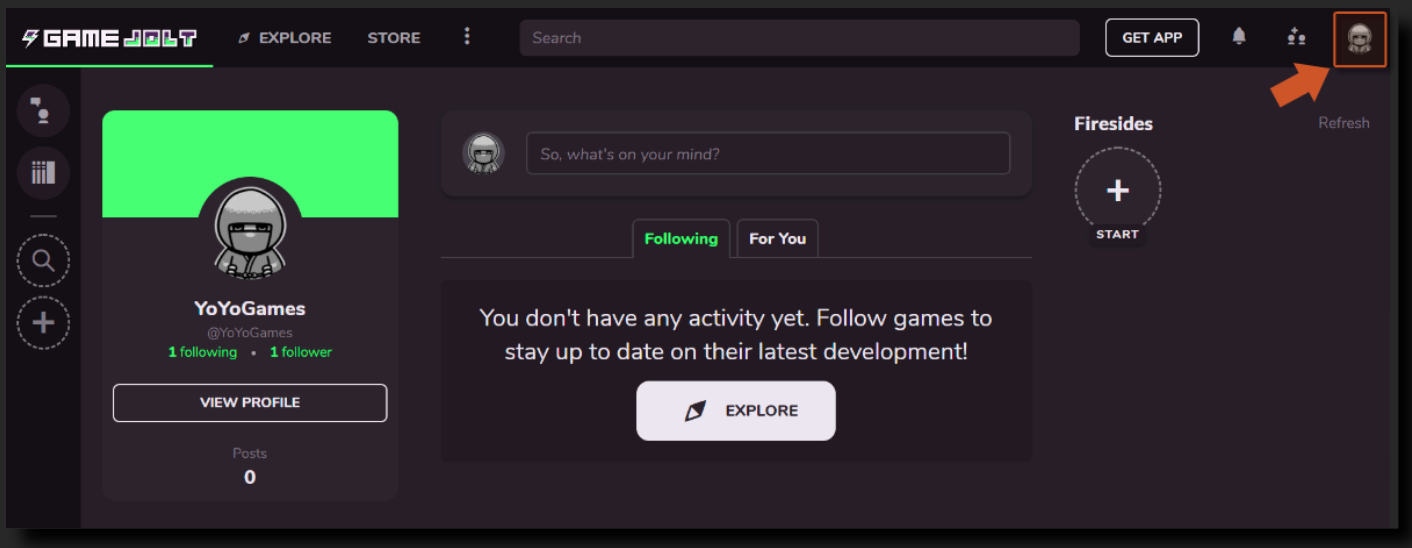
NOTE This API uses structs as a way to maintain and organize data that is returned by the GameJolt Server. Please check the **Structs Overview** section for more information. The structs' information will also be referenced inside the respective methods that use/return that given struct.

GameJolt Setup

This section will describe the steps for creating your first game using GameJolt webpage.

INFO Make sure you have, first of all, created an account.

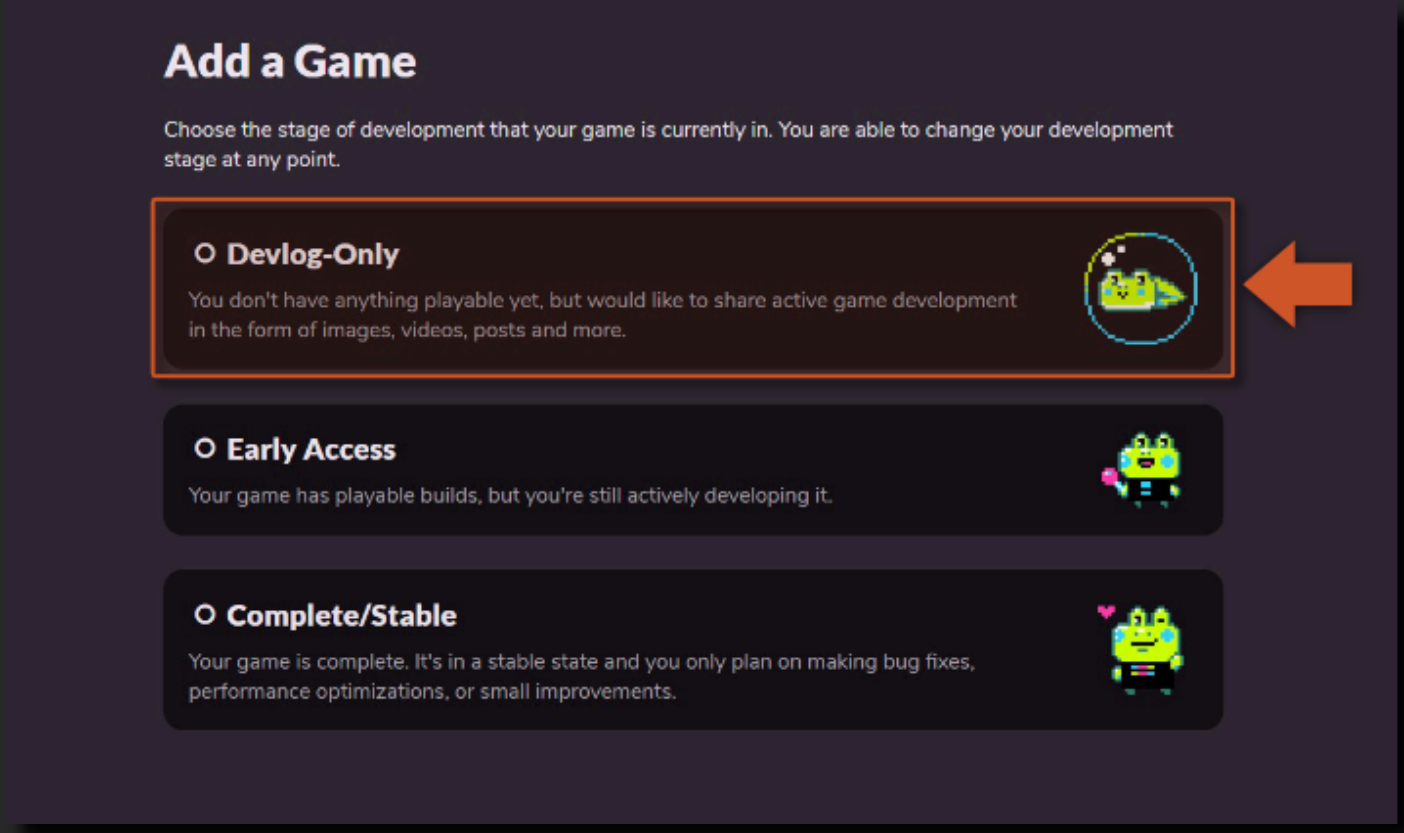
1. To create a new game click on your profile picture (on the top left corner of the screen):



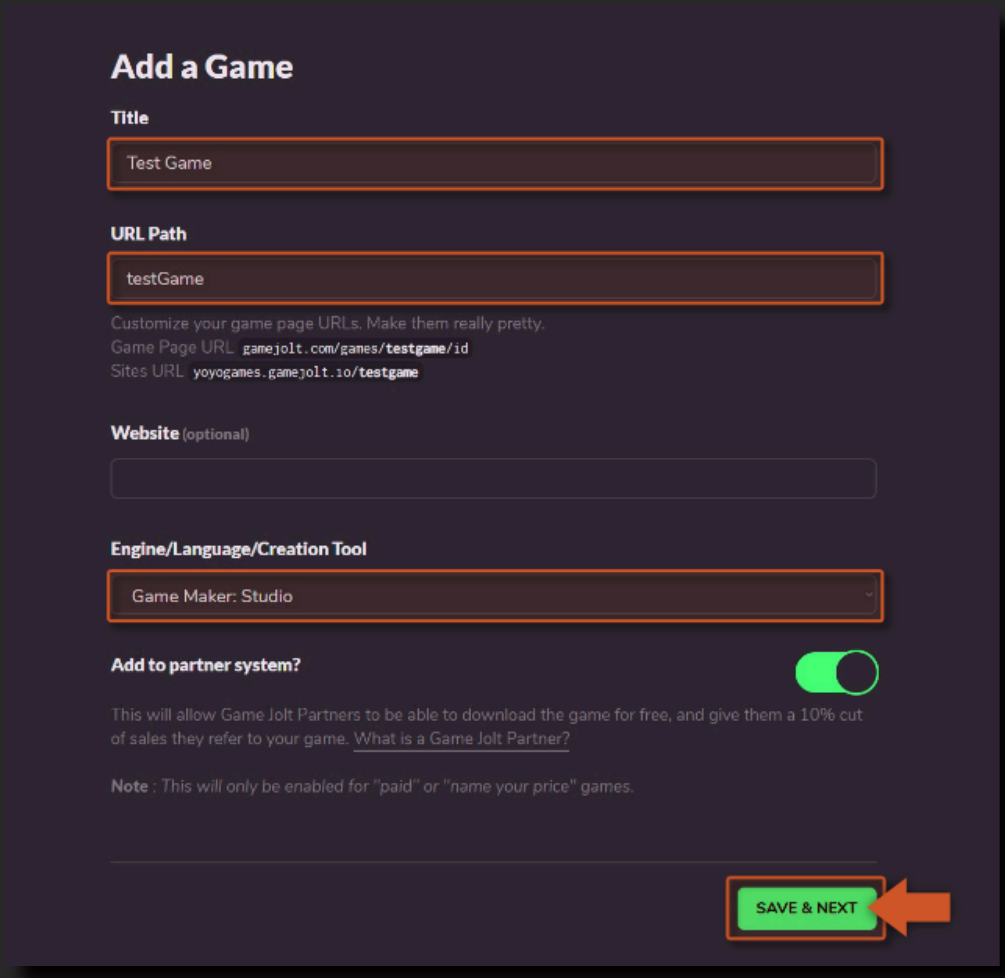
2. Select **Add a Game** from the drop down menu.

IMPORTANT If that option is NOT available visit this link instead.

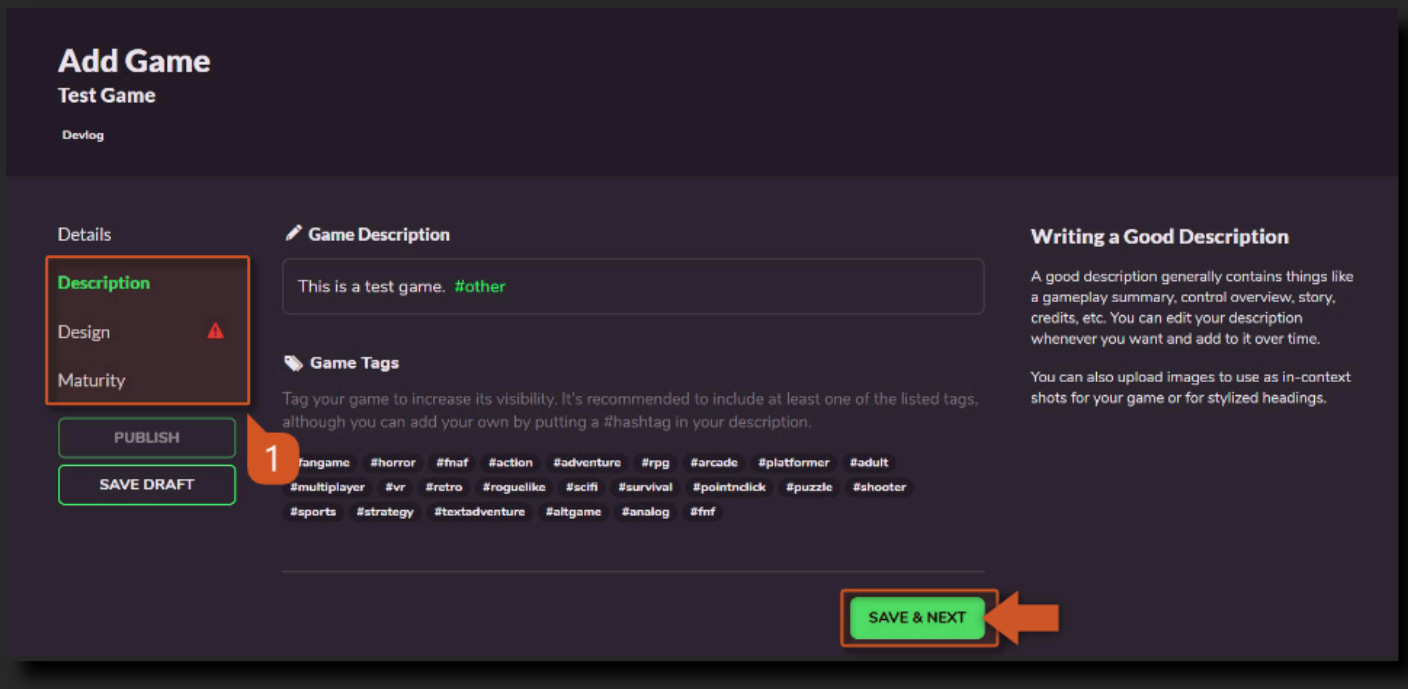
3. For the purpose of this tutorial we don't have a game yet so we will start by creating a development blog:



4. Now we can fill in the basic information about the game: **Title**, **URL Path** and **Engine**. After that information is filled just click the **Save & Next** button at the end of the form:

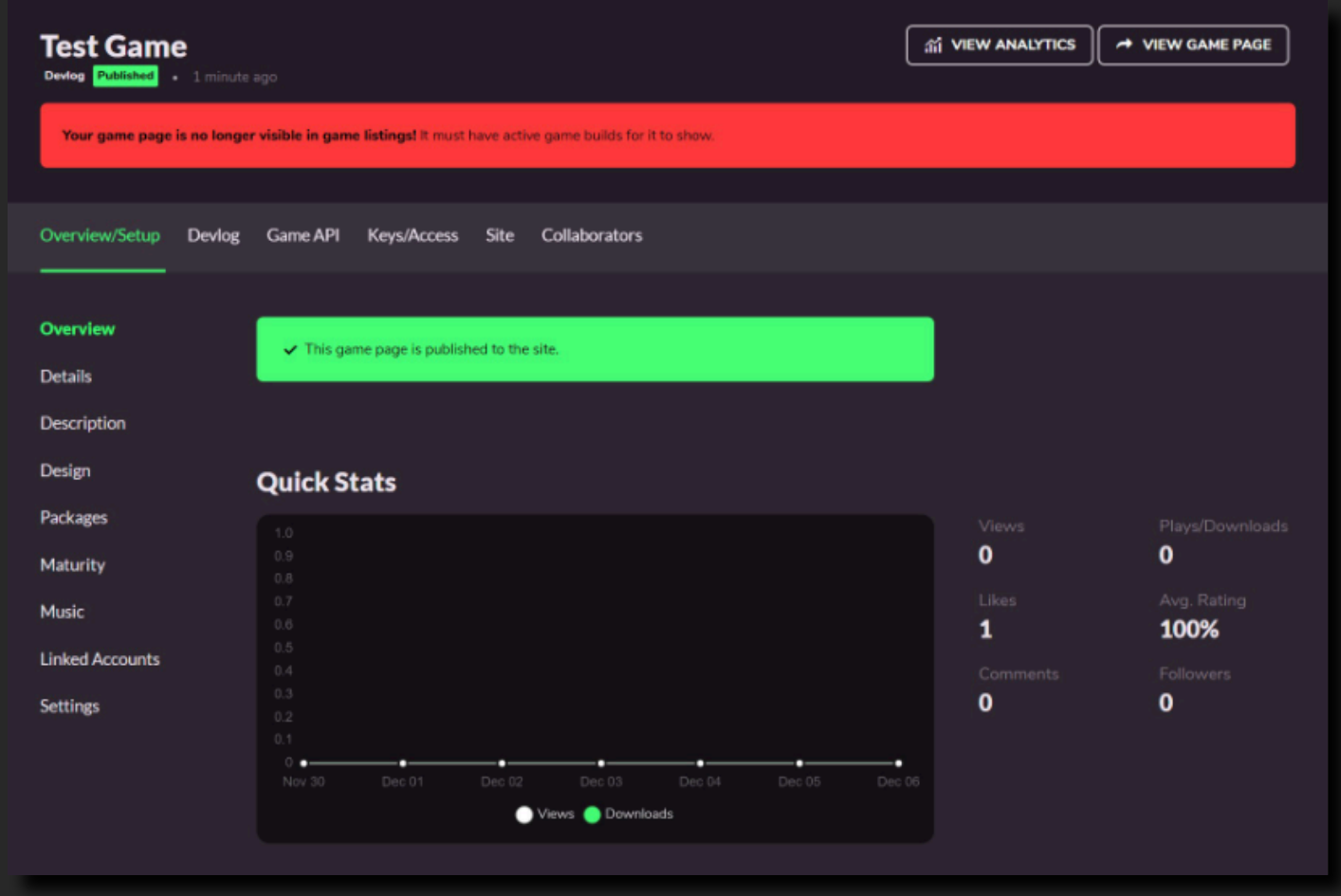


5. You should now have your project, but we are not done yet we need to go through every topic on the left side menu **(1)** and provide the required metadata information for our game:



IMPORTANT Don't forget to click **Save & Next** on each page to save the settings.

6. After everything is filled we should now be able to press the green **Publish** button to finish publishing the game:



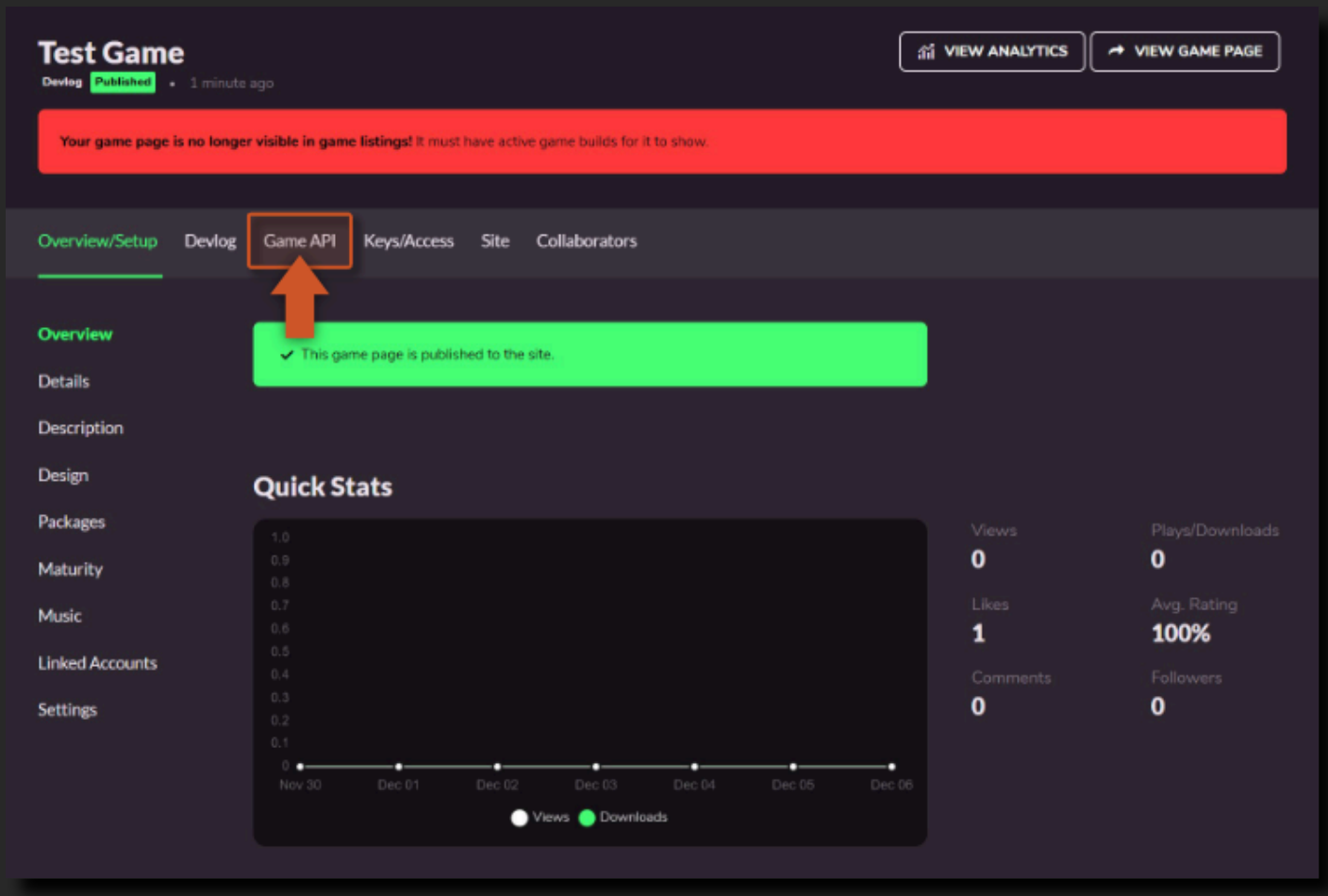
7. We should now have a published game ready to use with the **GameJolt** extension.

Project Setup

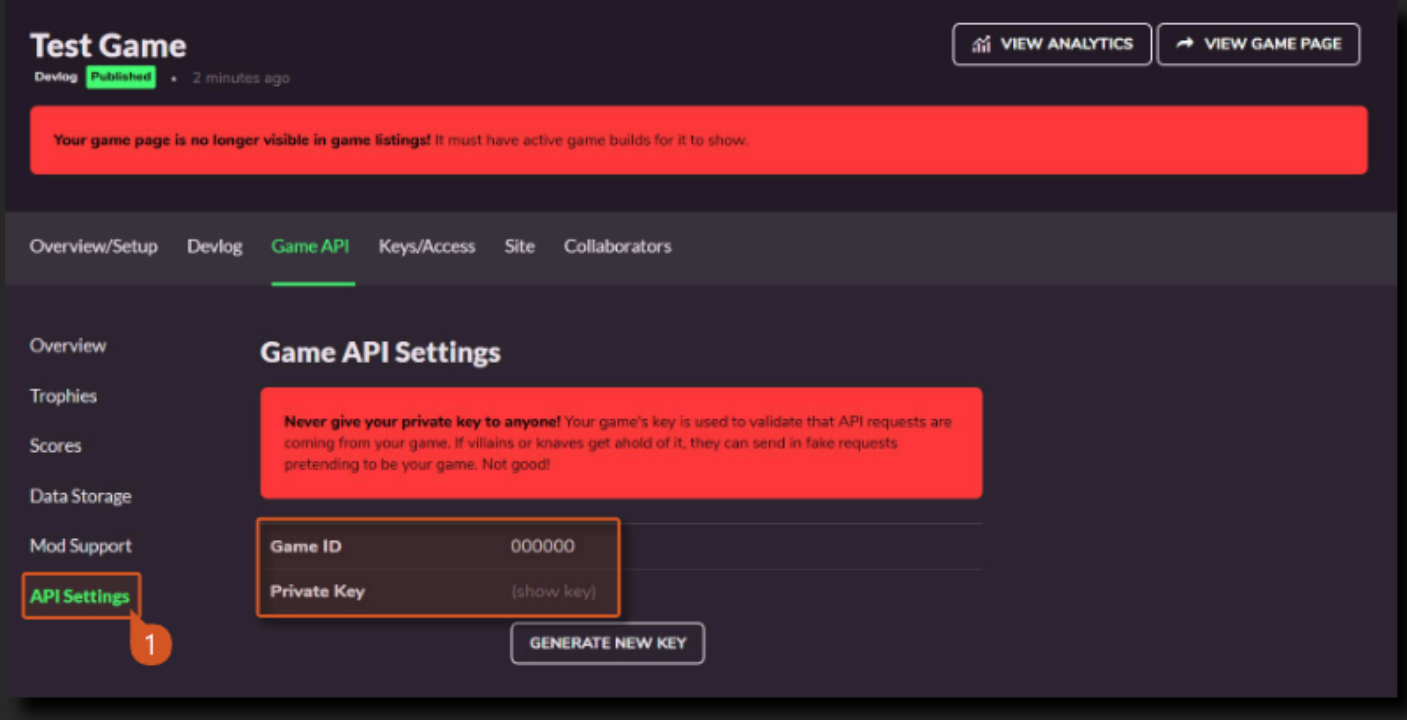
We will now setup the GameMaker project to start using the extension.

INFO Make sure you already have a game or have followed the GameJolt Setup guide first.

1. We need to go into our project management page and select **Game API** from the top menu:

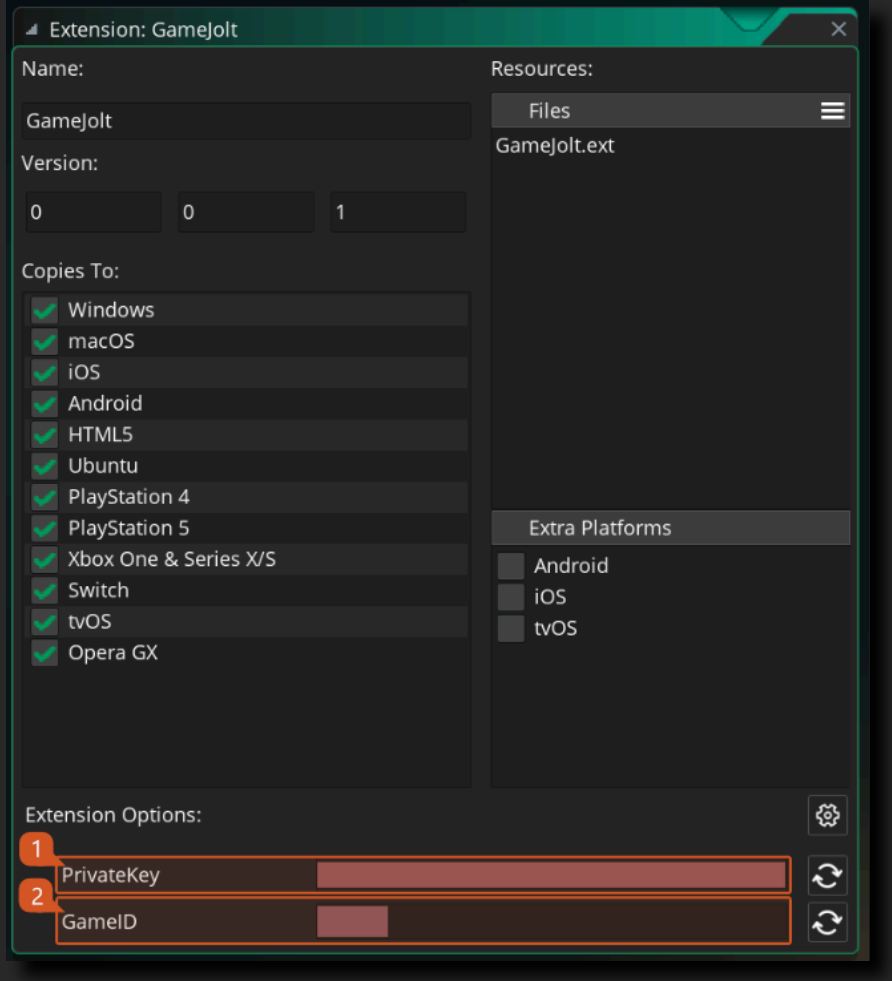


2. On this new page click the **API Settings** panel on the left menu. Here you will find both your **Game ID** and **Private Key** that are required for finishing the extension setup.



IMPORTANT This information should never be shared with anyone. If at some point in time you think it might have been compromised you can and should press the **Generate New Key** button to generate a new one (you will need to update your GameMaker Studio project as well).

3. Locate the **GameJolt** extension file within the extension project and double click it to open the extension window. On the bottom of that window you can now fill in the required data for your game: the **Game ID** (1) and the **Private Key** (2) values, obtained above.



4. Done, you are now ready to start using the **GameJolt API** extension.

Could Data Storage

Sync saved games, user-created levels, debug logs save any bit of data to the player's account, or globally for your game. The clouds are the limit!

Functions

The following functions are provided for working with cloud data storage:

- GameJolt_DataStorage_Fetch
- GameJolt_DataStorage_Fetch_Global
- GameJolt_DataStorage_GetKeys
- GameJolt_DataStorage_GetKeys_Global
- GameJolt_DataStorage_Remove
- GameJolt_DataStorage_Remove_Global
- GameJolt_DataStorage_Set
- GameJolt_DataStorage_Set_Global
- GameJolt_DataStorage_Update
- GameJolt_DataStorage_Update_Global

Other

Extra information regarding the data storage functions:

- Operations Table

GameJolt_DataStorage_Fetch

This function fetches data from the user data storage.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Fetch(key, [callback_success], [callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
callback_success	method	The callback function executed if the request succeeds (value is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Fetch("heroLevel",
    function(data)
    {
        heroLevel = real(data);
    },
    function(message)
    {
        show_message_async(message)
    });
```

In the code sample above we perform a fetch action to retrieve the **user data** stored with the key `"heroLevel"`, we then provide two methods the success method will display the

value associated with that key and the failure method will show a message with the error information.

GameJolt_DataStorage_Fetch_Global

This function fetches data from the global data storage.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Fetch_Global(key, [callback_success], [callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
callback_success	method	The callback function executed if the request succeeds (value is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Fetch_Global("lastBossKillCount",
    function(data)
    {
        bossKillCount = real(data);
    },
    function(message)
    {
        show_message_async(message)
    });
```

In the code sample above we perform a fetch action to retrieve the **global data** stored with the key `"lastBossKillCount"`, we then provide two methods the success method will display

the value associated with that key and the failure method will show a message with the error information.

GameJolt_DataStorage_GetKeys

This function fetches keys of data items from the user data storage.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_GetKeys(pattern, [callback_success], [callback_failed])
```

Argument	Type	Description
pattern	string	The pattern to apply when querying existing keys
callback_success	method	The callback function executed if the request succeeds (arrays of strings is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_GetKeys("*",
    function(array)
    {
        show_message_async(array);
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform a fetch action to retrieve the keys from the **user data storage**, that follow the pattern `"*"` (meaning **all**). We then provide two methods: the success method will display the array of existing keys and the failure method will show a message with the error information.

GameJolt_DataStorage_GetKeys_Global

This function fetches keys of data items from the global data storage.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_GetKeys_Global(pattern, [callback_success], [callback_failed])
```

Argument	Type	Description
pattern	string	The pattern to apply when querying existing keys
callback_success	method	The callback function executed if the request succeeds (arrays of strings is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_GetKeys_Global("*",
    function(array)
    {
        show_message_async(array);
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform a fetch action to retrieve the keys from the **global data storage**, that follow the pattern `"*"` (meaning **all**). We then provide two methods: the success method will display the array of existing keys and the failure method will show a message with the error information.

GameJolt_DataStorage_Remove

This function removes data items from the user data store.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Remove(key, [callback_success], [callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
callback_success	method	The callback function executed if the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Remove("hasArrows",
    function()
    {
        show_message_async("Success!!");
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform an action to remove a key (`"hasArrows"`) from the **user data storage**. We then provide two methods that will display the success of failure of the task.

GameJolt_DataStorage_Remove_Global

This function removes data items from the global data store.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Remove_Global(key, [callback_success], [callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
callback_success	method	The callback function executed if the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Remove("gameoverCount",
    function()
    {
        show_message_async("Success!!");
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform an action to remove a key (`"gameoverCount"`) from the **global data storage**. We then provide two methods that will display the success of failure of the task.

GameJolt_DataStorage_Set

This function sets data in the user data store.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Set(key, data, [callback_success], [callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
data	real/ string/ struct	The data value to associate with the provided key
callback_success	method	The callback function executed if the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Set("name", "HERO"
    function()
    {
        show_message_async("Success!!");
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform an action to set a key (`"name"`) from the **user data storage** to a specific value (`"HERO"`). We then provide two methods that will display the success of failure of the task.

GameJolt_DataStorage_Set_Global

This function sets data in the global data store.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Set_Global(key, data, [callback_success], [callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
data	real/ string/ struct	The data value to associate with the provided key
callback_success	method	The callback function executed if the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Set_Global("gameoverCount", 2
    function()
    {
        show_message_async("Success!!");
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform an action to set a key (`"gameoverCount"`) from the **global data storage** to a specific value (`2`). We then provide two methods that will display the success of failure of the task.

GameJolt_DataStorage_Update

This function updates user data in the data store applying a selected operation.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Update(key, data, operation, [callback_success],
[callback_failed])
```

Argument	Type	Description
key	string	The identifier key of the data
data	real/ string/ struct	The data value used by the operation
operation	string	The operation to be performed with the provided data (see **Operations Table**)
callback_success	method	The callback function executed if the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Update("currentLevel", 1, "add"
    function()
    {
        show_message_async("Success!!");
    },
    function(message)
    {
```

```
            show_message_async(message)
        }
    )
```

In the code sample above we perform an action to update a key (`"currentLevel"`) from the **user data storage** using the `"add"` operation and passing a given amount `1`. We then provide two methods that will display the success of failure of the task.

GameJolt_DataStorage_Update_Global

This function updates global data in the data store applying a selected operation.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_DataStorage_Update_Global(key, data, operation,  [callback_success], [callback_failed])
```

Argument	Tyoe	Description
key	string	The identifier key of the data
data	real/ string/ struct	The data value used by the operation
operation	string	The operation to be performed with the provided data (see **Operations Table**)
callback_success	method	The callback function executed if the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_DataStorage_Update("gameoverCount", 1, "add"
    function()
    {
        show_message_async("Success!!");
    },
    function(message)
    {
```

```
            show_message_async(message)
        }
    )
```

In the code sample above we perform an action to update a key (`"gameoverCount"`) from the **global data storage** using the `"add"` operation and passing a given amount `1` . We then provide two methods that will display the success of failure of the task.

Operations Table

This is a table of operations that the developer can perform on data stored inside the GameJolt cloud storage system. These operation are to be used with the functions:

- GameJolt_DataStorage_Update
- GameJolt_DataStorage_Update_Global

Operation	Description
`"add"`	Adds the `value` to the current data store item (reals only).
`"subtract"`	Subtracts the `value` from the current data store item (reals only).
`"multiply"`	Multiplies the `value` by the current data store item (reals only).
`"divide"`	Divides the current data store item by the `value` (reals only).
`"append"`	Appends the `value` to the current data store item.
`"prepend"`	Prepends the `value` to the current data store item.

Friends

Because playing with friends is even better and thrilling. List all user's friends and create an healthy competition environment to keep them all engaged.

Functions

The following function is provided for fetching friends:

- GameJolt_Friends

GameJolt_Friends

This function queries an array of `user_id` represent the a user's friends (further information can be queried using the GameJolt_User_FetchWithUserID).

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Friends([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed if the request succeeds (array of `user_id` is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Friends(
    function(array)
    {
        // If the array has at least one entry
        if (array_length(array) > 1)
        {
            GameJolt_User_FetchWithUserID(array[0], function(data) {
                show_message_async(data);
            })
        }
    }
);
```

In the code sample above we perform an action to retrieve an array with the `user_id`s of all the logged user friends. We then provide a method that will fetch the user information for the first user id in the returned array (using the GameJolt_User_FetchWithUserID function), printing out the result when finished.

Scores

Implement leaderboards in your game to allow anyone to battle it out for the top spots. You create the leaderboards, you control the scoring. You can even allow guests to score without a Game Jolt account.

Functions

The following functions are provided for working with scores and leaderboards:

- GameJolt_Scores_Add
- GameJolt_Scores_Add_Guest
- GameJolt_Scores_Fetch
- GameJolt_Scores_Fetch_BetterThan
- GameJolt_Scores_Fetch_Guest
- GameJolt_Scores_Fetch_Me
- GameJolt_Scores_Fetch_WorseThan
- GameJolt_Scores_Rank
- GameJolt_Scores_Tables

Structs

Extra details on structs that are returned as score fetching results:

- ScoreData

GameJolt_Scores_Add

This function adds a score representing the logged user.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Add(table_id, score, sort, extra_data, [callback_success],
[callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table to submit to.
score	string	This is a string value associated with the score.
sort	real	This is a numerical sorting value associated with the score. All sorting will be based on this number.
extra_data	string	If there's any extra data you would like to store as a string, you can use this variable.
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Add(668889,"Points: " + string(points), points, "this is some
metadata",
    function()
    {
```

```
    show_message_async("Score submitted successfully!")
});
```

The code above will submit a score value for the current logged user to a given `table_id`, and print a success message if it succeeds.

GameJolt_Scores_Add_Guest

This function adds a score representing a guest user.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Add_Guest(guest, table_id, score, sort,
extra_data, [callback_success], [callback_failed])
```

Argument	Type	Description
guest	string	The guest's name.
table_id	real	The ID of the score table to submit to.
score	string	This is a string value associated with the score.
sort	real	This is a numerical sorting value associated with the score. All sorting will be based on this number.
extra_data	string	If there's any extra data you would like to store as a string, you can use this variable.
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Add_Guest("guestName123", 668889, "Points: " + string(points),
points, "this is some metadata",
    function()
    {
        show_message_async("Score submitted successfully!")
    });
```

The code above will submit a score value for a guest user (`"guestName123"`) to a given `table_id`, and print a success message if it succeeds.

GameJolt_Scores_Fetch

This function fetches scores from a score table.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Fetch(table_id, limit, [callback_success], [callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table to submit to.
limit	real	The number of scores you'd like to return.
callback_success	method	The callback function executed if the request succeeds (an array of ScoreData structs is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Fetch(668889, 10, function(scoresArray) {

    var scoreData = scoresArray[0];

    var _score = scoreData.score;
    var sort = scoreData.sort;
    var extra_data = scoreData.extra_data;
    var user = scoreData.user;
    var user_id = scoreData.user_id;
    var guest = scoreData.guest;
    var stored = scoreData.stored;
    var stored_timestamp = scoreData.stored_timestamp;
```

```
})
```

The code above will fetch a total of `10` scores from a given `table_id` and on success collect all the data that is presented from the first entry in the returned array.

GameJolt_Scores_Fetch_BetterThan

This function fetches score from a score table, that are better than a given value.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Fetch_BetterThan(table_id, better_than, limit, [callback_success], [callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table to submit to.
better_than	real	Fetch only scores better than this score sort value.
limit	real	The number of scores you'd like to return.
callback_success	method	The callback function executed if the request succeeds (an array of ScoreData structs is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Fetch_BetterThan(668889, 10, 9999, function(scoresArray) {

    var scoreData = scoresArray[0];

    var _score = scoreData.score;
    var sort = scoreData.sort;
    var extra_data = scoreData.extra_data;
    var user = scoreData.user;
    var user_id = scoreData.user_id;
    var guest = scoreData.guest;
```

```
    var stored = scoreData.stored;
    var stored_timestamp = scoreData.stored_timestamp;

})
```

The code above will fetch a total of `10` scores from a given `table_id` that are better than the value `9999` and on success collect all the data that is presented from the first entry in the returned array.

GameJolt_Scores_Fetch_Guest

This function fetches the score of guest user from a score table.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Fetch_Guest(table_id, guest, [callback_success], [callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table
callback_success	method	The callback function executed if the request succeeds (a ScoreData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Fetch_Guest(668889, "guestName123", function(scoreData) {

    var _score = scoreData.score;
    var sort = scoreData.sort;
    var extra_data = scoreData.extra_data;
    var user = scoreData.user;
    var user_id = scoreData.user_id;
    var guest = scoreData.guest;
    var stored = scoreData.stored;
    var stored_timestamp = scoreData.stored_timestamp;

})
```

The code above will fetch a **ScoreData** struct from the given `table_id` that is associated with the guest `"guestName123"` and on success collect all the data that is presented in the struct.

GameJolt_Scores_Fetch_Me

This function fetches the score of the current user from a score table.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Fetch_Me(table_id, [callback_success], [callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table
callback_success	method	The callback function executed if the request succeeds (a ScoreData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Fetch_Guest(668889, function(scoreData) {

    var _score = scoreData.score;
    var sort = scoreData.sort;
    var extra_data = scoreData.extra_data;
    var user = scoreData.user;
    var user_id = scoreData.user_id;
    var guest = scoreData.guest;
    var stored = scoreData.stored;
    var stored_timestamp = scoreData.stored_timestamp;

})
```

The code above will fetch a **ScoreData** struct from the given `table_id` that is associated with the currently logged user and on success collect all the data that is presented in the struct.

GameJolt_Scores_Fetch_WorseThan

This function fetches score from a score table, that are worse than a given value.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Fetch_WorseThan(table_id, worse_than, limit, [callback_success], [callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table to submit to.
worse_than	real	Fetch only scores worse than this score sort value.
limit	real	The number of scores you'd like to return.
callback_success	method	The callback function executed if the request succeeds (an array of ScoreData structs is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Fetch_WorseThan(668889, 10, 10, function(scoresArray) {

    var scoreData = scoresArray[0];

    var _score = scoreData.score;
    var sort = scoreData.sort;
    var extra_data = scoreData.extra_data;
    var user = scoreData.user;
    var user_id = scoreData.user_id;
    var guest = scoreData.guest;
```

```
    var stored = scoreData.stored;
    var stored_timestamp = scoreData.stored_timestamp;

})
```

The code above will fetch a total of `10` scores from a given `table_id` that are worse than the value `10` and on success collect all the data that is presented from the first entry in the returned array.

GameJolt_Scores_Rank

This functions gets a rank for a specific score.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Rank(table_id, sort, [callback_success], [callback_failed])
```

Argument	Type	Description
table_id	real	The ID of the score table to submit to.
sort	real	This is a numerical sorting value associated with the score. All sorting will be based on this number.
callback_success	method	The callback function executed if the request succeeds (a rank value is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Rank(668889, 100,
    function(rank) {
        show_message_async(rank)
    });
```

The code above will query the rank on the current logged user on the given `table_id` and show it to the user.

GameJolt_Scores_Tables

This function fetches a list of score tables.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Scores_Tables([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed if the request succeeds (array of `table_id` is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Scores_Tables(function(array)
    {
        show_message_async(array);
    },
    function(message)
    {
        show_message_async(message)
    }
)
```

In the code sample above we perform a fetch action to retrieve all the score `table_id`s associated with this game. We then provide two methods: the success method will display the array of existing table ids and the failure method will show a message with the error information.

Sessions

Track when and how long each player is active in your game. You can then view stats such as avg. play time per session, total time played across users, and even see how many people are playing your game in real time.

Functions

The following functions are provided for working with sessions:

- GameJolt_Session_Check
- GameJolt_Session_Close
- GameJolt_Session_Open
- GameJolt_Session_Ping_Active
- GameJolt_Session_Ping_Idle

GameJolt_Session_Check

This function checks to see if there is an open session for the user. Can be used to see if a particular user account is active in the game.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

NOTE This function calls `callback_failed` when no open session exists. This behavior is different from other functions which use the callback to indicate an error state.

Syntax:

```
GameJolt_Session_Check([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if no session is open OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Session_Check(
    function()
    {
        show_message_async("Session is open")
    },
    function()
    {
        GameJolt_Session_Open();
    });
```

The code sample above will check if there is a currently open session and if there is shows the message `"Session Connected"`, otherwise attemps to open a new session (using the method GameJolt_Session_Open).

GameJolt_Session_Close

This function closes the active session.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Session_Close([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Session_Close(undefined, function() { show_debug_message("Session Close
Failed"); });
```

The code sample above tries to close a session and provides a debug message if the session fails to close.

GameJolt_Session_Open

This function opens a game session for a particular user and allows you to tell Game Jolt that a user is playing your game. You must ping the session (using the functions GameJolt_Session_Ping_Active or GameJolt_Session_Ping_Idle) to keep it active and you must close it (using the GameJolt_Session_Close) when you're done with it.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Session_Open([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Session_Open(
    function()
    {
        show_message_async("Session Opened!!")
    },
    function(message)
    {
        show_message_async(message)
    });
```

The code sample above tries to open a new session and provides a message with the regarding its success.

GameJolt_Session_Ping_Active

This function pings an open session to tell the system that the session is still being used (**Active**). If the session hasn't been pinged within 120 seconds, the system will close the session and you will have to open another one. It's recommended that you ping about every 30 seconds or so to keep the system from clearing out your session.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

TIP You can also let the system know the player is idled, using GameJolt_Session_Ping_Idle.

Syntax:

```
GameJolt_Session_Ping_Active([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Session_Ping_Active(undefined,
    function(message)
    {
        show_message_async("Unable to ping");
    });
```

The code sample above tries to ping to the current session with an **active** user state and shows a message if unable to ping.

GameJolt_Session_Ping_Idle

This function pings an open session to tell the system that the session is still being used (**Idle**). If the session hasn't been pinged within 120 seconds, the system will close the session and you will have to open another one. It's recommended that you ping about every 30 seconds or so to keep the system from clearing out your session.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

TIP You can also let the system know the player is active, using GameJolt_Session_Ping_Active.

Syntax:

```
GameJolt_Session_Ping_Idle([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Session_Ping_Idle(undefined,
    function(message)
    {
        show_message_async("Unable to ping");
    });
```

The code sample above tries to ping to the current session with an **idle** user state and shows a message if unable to ping.

Time

Never miss the beat with a simple query to the server for precise time information.

Functions

The following function is provided for working with time:

- GameJolt_Time

Structs

Extra details on structs that are returned as time fetching results:

- TimeData

GameJolt_Time

This function queries the time of the Game Jolt server.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Time([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed if the request succeeds (a TimeData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Time(function(timeData)
    {
        var _timestamp = timeData.timestamp;
        var _timezone = timeData.timezone;
        var _year = timeData.year;
        var _month = timeData.month;
        var _day = timeData.day;
        var _hour = timeData.hour;
        var _minute = timeData.minute;
        var _second = timeData.second;

        if (timeData.hour > 0 && timeDate.hour < 3)
        {
            show_debug_message("It's too late, time to go to sleep!");
        }
    });
```

The code sample above queries the time for the Game Jolt server and if it is too late show a sleeping message.

Trophies

Feed into your player base's hunger for trophy hunting. Trophies will sync to their Game Jolt profile for all to see as badges of honor.

Functions

The following functions are provided for working with trophies:

- GameJolt_Trophies_Fetch
- GameJolt_Trophies_Fetch_All
- GameJolt_Trophies_Remove
- GameJolt_Trophies_Update

Structs

Extra details on structs that are returned as trophy fetching results:

- TrophyData

GameJolt_Trophies_Fetch

This function fetches a trophy data (see TrophyData).

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Trophies_Fetch(trophy_id, [callback_success], [callback_failed])
```

Argument	Type	Description
trophy_id	real	The trophy identifier
callback_success	method	The callback function executed if the request succeeds (a TrophyData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Trophies_Fetch(myTropyID,
    function(_trophyData)
    {
        var _id = _trophyData.id;
        var _title = _trophyData.title;
        var _description = _trophyData.description;
        var _difficulty = _trophyData.difficulty;
        var _image_url = _trophyData.image_url;
        var _achieved = _trophyData.achieved;

        show_debug_message("Trophy info: " + string(_trophyData));
    });
```

The code sample above will fetch a single trophy data (see TrophyData) and print it to the debugger if the task succeeds.

GameJolt_Trophies_Fetch_All

This function fetches an array of trophy data (see TrophyData).

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Trophies_Fetch_All(achieved, [callback_success], [callback_failed])
```

Argument	Type	Description
achieved	boolean	Whether or not to fetch only achieved trophies
callback_success	method	The callback function executed if the request succeeds (array of TrophyData structs is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_Trophies_Fetch_All(true,
    function(trophiesArray)
    {
        var _trophyData = trophiesArray[0];

        var _id = _trophyData.id;
        var _title = _trophyData.title;
        var _description = _trophyData.description;
        var _difficulty = _trophyData.difficulty;
        var _image_url = _trophyData.image_url;
        var _achieved = _trophyData.achieved;
```

```
        show_debug_message("Trophy info: " + string(trophiesArray));
    });
```

The code sample above will fetch all achieved trophies (see **TrophyData**) and print it to the debugger if the task succeeds.

GameJolt_Trophies_Remove

This function removes a previously achieved trophy for a particular user.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Trophies_Remove(trophy_id, [callback_success], [callback_failed])
```

Argument	Type	Description
trophy_id	string	The trophy identifier
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
if (cheating == true)
    GameJolt_Trophies_Remove(trophyID, function() { show_message_async("You were
caught cheating!"); });
```

The code sample above checks if the variable cheating is `true` and if so it removes the current trophy from the user and prints a message that he was caught cheating.

GameJolt_Trophies_Update

This function sets a trophy as achieved for a particular user.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_Trophies_Update(trophy_id, [callback_success], [callback_failed])
```

gument	Type	Description
trophy_id	string	The trophy identifier
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
if (completeGame == true)
    GameJolt_Trophies_Update(trophyID, function() { show_message_async("You reached
the end of the game!"); });
```

The code sample above checks if the user completed tjat game it updates the current trophy and prints a congratulations message.

Users

Log in, logout and log in from previously cached user data. Access user's account information to customize your game to your target audience.

Functions

The following functions are provided for working with user and accounts:

- GameJolt_User_FetchMe
- GameJolt_User_FetchWithUserID
- GameJolt_User_FetchWithUserName
- GameJolt_User_IsLogged
- GameJolt_User_LogIn
- GameJolt_User_LogIn_FromCache
- GameJolt_User_LogOut

Structs

Extra details on structs that are returned as user fetching results:

- UserData

GameJolt_User_FetchMe

This function fetches the logged user data (see UserData).

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_User_FetchMe([callback_success], [callback_failed])
```

Argument	Type	Description
callback_success	method	The callback function executed if the request succeeds (a UserData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_User_FetchMe(
    function(_userData)
    {
        var _id = _userData.id;
        var _type = _userData.type;
        var _username = _userData.username;
        var _avatar_url = _userData.avatar_url;
        var _signed_up = _userData.signed_up;
        var _signed_up_timestamp = _userData.signed_up_timestamp;
        var _last_logged_in = _userData.last_logged_in;
        var _last_logged_in_timestamp = _userData.last_logged_in_timestamp;
        var _status = _userData.status;
        var _developer_name = _userData.developer_name;
        var _developer_website = _userData.developer_website;
        var _developer_description = _userData.developer_description;

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

```
function(message)
{
    show_message_async(message)
});
```

The code above will fetch the current logged user data (see **UserData**) and after successfully retrieving it it starts loading the `avatar_url` of the user (using the function **sprite_add**) if it fails it prints an asynchronous message with the error.

GameJolt_User_FetchWithUserID

This function fetches the user data of a specific user id (see UserData).

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

TIP This function is helpful for retrieving information from friends `user_id`, see GameJolt_Friends.

Syntax:

```
GameJolt_User_FetchWithUserID(user_id, [callback_success], [callback_failed])
```

Argument	Type	Description
user_id	string	The player's user id
callback_success	method	The callback function executed if the request succeeds (a UserData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

N/A

Example:

```
GameJolt_User_FetchWithUserID(32323,
    function(_userData)
    {
        var _id = _userData.id;
        var _type = _userData.type;
        var _username = _userData.username;
        var _avatar_url = _userData.avatar_url;
        var _signed_up = _userData.signed_up;
        var _signed_up_timestamp = _userData.signed_up_timestamp;
        var _last_logged_in = _userData.last_logged_in;
```

```
        var _last_logged_in_timestamp = _userData.last_logged_in_timestamp;
        var _status = _userData.status;
        var _developer_name = _userData.developer_name;
        var _developer_website = _userData.developer_website;
        var _developer_description = _userData.developer_description;

        sprite = sprite_add(_avatar_url, 0, 0, 0, 0, 0);
    },
    function(message)
    {
        show_message_async(message)
    });
```

The code above will fetch the user data (see **UserData**) of a given user id (you can get the user ids of friends using the **GameJolt_Friends** function) and after successfully retrieving it it starts loading the `avatar_url` of the user (using the function **sprite_add**) if it fails it prints an asynchronous message with the error.

GameJolt_User_FetchWithUserName

This function fetches the user data of a specific username (see UserData).

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

Syntax:

```
GameJolt_User_FetchWithUserName(username, [callback_success], [callback_failed])
```

Argument	Type	Description
username	string	The player's username
callback_success	method	The callback function executed if the request succeeds (a UserData struct is passed as argument) OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

N/A

Example:

```
GameJolt_User_FetchWithUserID("gameMaster",
    function(_userData)
    {
        var _id = _userData.id;
        var _type = _userData.type;
        var _username = _userData.username;
        var _avatar_url = _userData.avatar_url;
        var _signed_up = _userData.signed_up;
        var _signed_up_timestamp = _userData.signed_up_timestamp;
        var _last_logged_in = _userData.last_logged_in;
        var _last_logged_in_timestamp = _userData.last_logged_in_timestamp;
        var _status = _userData.status;
        var _developer_name = _userData.developer_name;
        var _developer_website = _userData.developer_website;
        var _developer_description = _userData.developer_description;

        sprite = sprite_add(_avatar_url, 0, 0, 0, 0, 0);
```

```
    },
    function(message)
    {
        show_message_async(message)
    });
```

The code above will fetch the user data (see **UserData**) of a given `username` and after successfully retrieving it it starts loading the `avatar_url` of the user (using the function **sprite_add**) if it fails it prints an asynchronous message with the error.

GameJolt_User_IsLogged

This function checks if the user is currently logged in.

Syntax:

```
GameJolt_User_IsLogged()
```

Returns:

```
Bool
```

Example:

```
if (GameJolt_User_IsLogged())
{
    GameJolt_User_LogOut();
}
else GameJolt_User_LogIn_FromCache();
```

The code sample above will check if the user is currently logged in and if so it logs the user out (using the function GameJolt_User_LogOut) otherwise it will log the user in with the information stored in cache (using the function GameJolt_User_LogIn_FromCache).

GameJolt_User_LogIn

This function authenticates the user's information. This should be done before you make any calls for the user, to make sure the user's credentials (username and token) are valid.

This is an asynchronous function that will trigger either the `callback_success` method (if task is successful) or the `callback_failed` method (if task fails).

NOTE You can use GameJolt_User_LogIn_FromCache to login with cached data over the next sessions.

Syntax:

```
GameJolt_User_LogIn(user_name, game_token, [callback_success], [callback_failed])
```

Argument	Type	Description
user_name	string	The user's username.
game_token	string	The user's token.
callback_success	method	The callback function executed when the request succeeds OPTIONAL
callback_failed	method	The callback function executed if the request fails (error message is passed as argument) OPTIONAL

Returns:

```
N/A
```

Example:

```
GameJolt_User_LogIn(user, token,
    function()
    {
        instance_create_depth(200, 250, 0, objPlayer)
    },
    function(message)
    {
```

```
        show_message_async("Error: " + message)
    });
```

The code sample above will login a user to a specific game using the game token for that game. After a successful login it creates an instance of `ObjPlayer` on the room (using the **instance_create_depth** function), otherwise it will display an error message.

GameJolt_User_LogIn_FromCache

This function logs in using the cached data from previous sessions (see GameJolt_User_LogIn).

Syntax:

```
GameJolt_User_LogIn_FromCache()
```

Returns:

```
Bool
```

Example:

```
if (GameJolt_User_LogIn_FromCache())
    instance_create_depth(200, 250, 0, objPlayer)
```

The code sample above will try to login a user to a specific game using data cached from a previous session. If the log in is successful it creates an instance of `objPlayer` inside the room (using the instance_create_depth function).

GameJolt_User_LogOut

This function closes a session and clears its cache.

Syntax:

```
GameJolt_User_LogOut()
```

Returns:

```
N/A
```

Example:

```
if (GameJolt_User_IsLogged())
    GameJolt_User_LogOut()
```

The code sample above will check if the user is currently logged in (using the GameJolt_User_IsLogged function) and if so it logs the user out.

Structs

Some of the API asynchronous callback responses return data in the form of structs. This sections aims to deliver detailed information on each of the structs used within the GameJolt context.

The following **structs** (structures) are used as return members of API function calls:

- ScoreData
- TimeData
- TrophyData
- UserData

ScoreData

This struct is returned as an async result of the call to the following API function calls:

- GameJolt_Scores_Fetch
- GameJolt_Scores_Fetch_BetterThan
- GameJolt_Scores_Fetch_Guest
- GameJolt_Scores_Fetch_Me
- GameJolt_Scores_Fetch_WorseThan

and it contains details that describe a score.

Property	Type	Description
score	string	The score string (example: `"234 Coins"`)
sort	real	The score's numerical sort value.
extra_data	string	Any extra data associated with the score (example: `"Level 2"`)
user	string	If this is a user score, this is the display name for the user.
user_id	real	If this is a user score, this is the user's ID.
guest	string	If this is a guest score, this is the guest's submitted name.
stored	string	Returns when the score was logged by the user (example: `"1 week ago"`)
stored_timestamp	real	Returns the timestamp (in seconds) of when the score was logged by the user.

TimeData

This struct is returned as an async result of the call to the following API function calls:

- GameJolt_Time

and it contains details that describe the current time.

Property	Type	Description
timestamp	real	The UNIX time stamp (in seconds) representing the server's time.
timezone	string	The timezone of the server (example: `"America/New_York"`)
year	real	The current year.
month	real	The current month.
day	real	The day of the month.
hour	real	The hour of the day.
minute	real	The minute of the hour.
second	real	The seconds of the minute.

TrophyData

This struct is returned as an async result of the call to the following API function calls:

- GameJolt_Trophies_Fetch
- GameJolt_Trophies_Fetch_All

and it contains details that describe a trophy.

Property	Type	Description
id	real	The ID of the trophy.
title	string	The title of the trophy on the site.
description	string	The trophy description text.
difficulty	string	Either `"Bronze"`, `"Silver"`, `"Gold"` or `"Platinum"`.
image_url	string	The URL of the trophy's thumbnail image.
achieved	boolean/ string	Date/time when the trophy was achieved by the user, or `false` they haven't achieved it yet.

UserData

This struct is returned as an async result of the call to the following API function calls:

- GameJolt_User_FetchMe
- GameJolt_User_FetchWithUserID
- GameJolt_User_FetchWithUserName

and it contains details that describe a user.

Property	Type	Description
id	real	The ID of the user.
type	string	The type of user. Can be `"User"`, `"Developer"`, `"Moderator"`, or `"Administrator"`.
username	string	The user's username.
avatar_url	string	The URL of the user's avatar.
signed_up	string	How long ago the user signed up (example: `"1 year ago"`)
signed_up_timestamp	real	The timestamp (in seconds) of when the user signed up.
last_logged_in	string	How long ago the user was last logged in. Will be `"Online Now"` if the user is currently online. (example: `"2 minutes ago"`)
last_logged_in_timestamp	real	The timestamp (in seconds) of when the user was last logged in.
status	string	`"Active"` if the user is still a member of the site. `"Banned"` if they've been banned.
developer_name	string	The user's display name.
developer_website	string	The user's website (or empty string if not specified)
developer_description	string	The user's profile markdown description.