Socket.io Client for HTML5

Made with <3 by Ivan Fonseca.

General Info

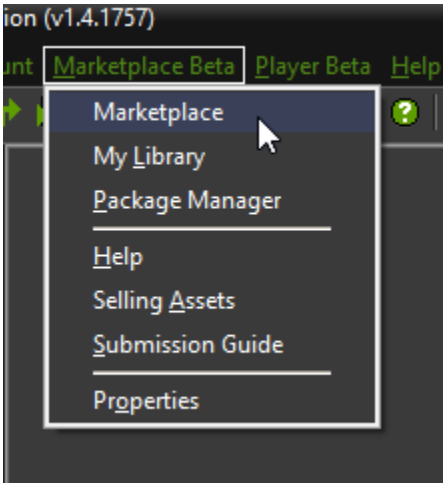
Socket.io is a real-time communication framework. It can be used for everything from web apps, multiplayer games and more. The server is written in Node.js, and the client is usually written in Javascript, but there are client libraries for just about every language now. This extension allows you to use Socket.io with the HTML5 export module in GameMaker: Studio.

Requirements

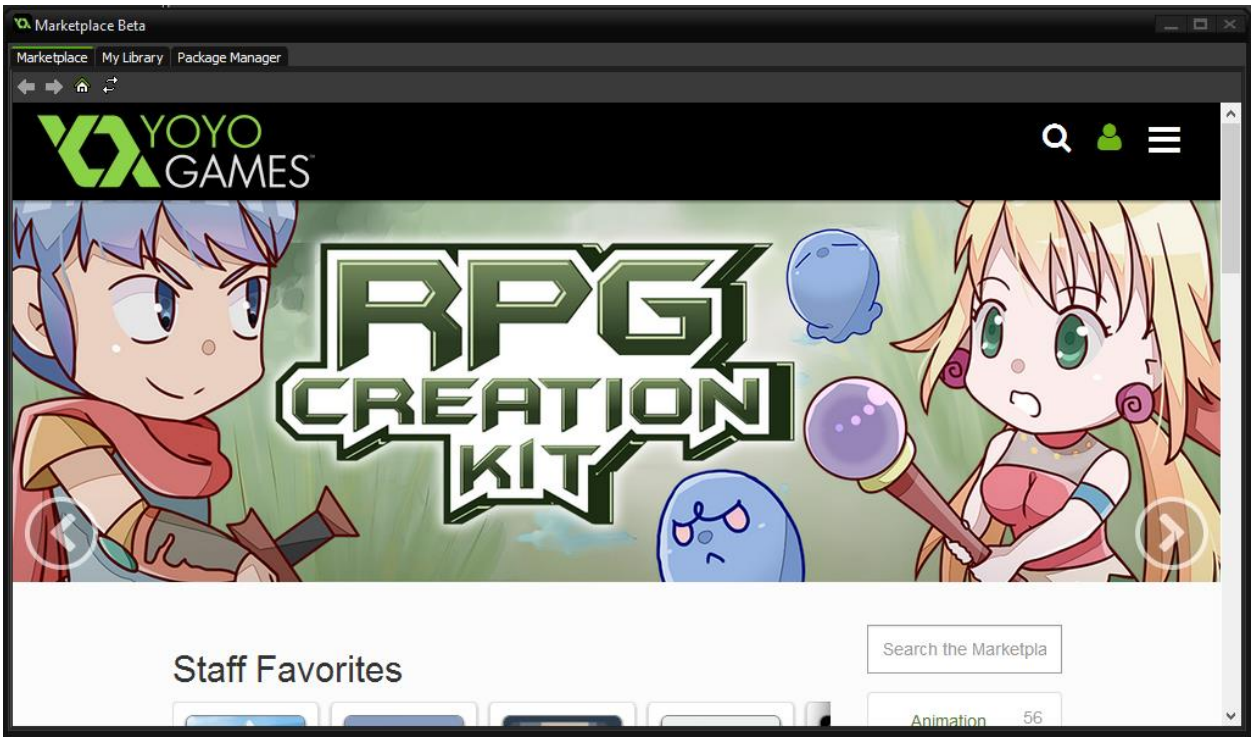
You must have the professional version of GameMaker: Studio and the HTML5 export module. This extension **will not work** with other exports.

Installation

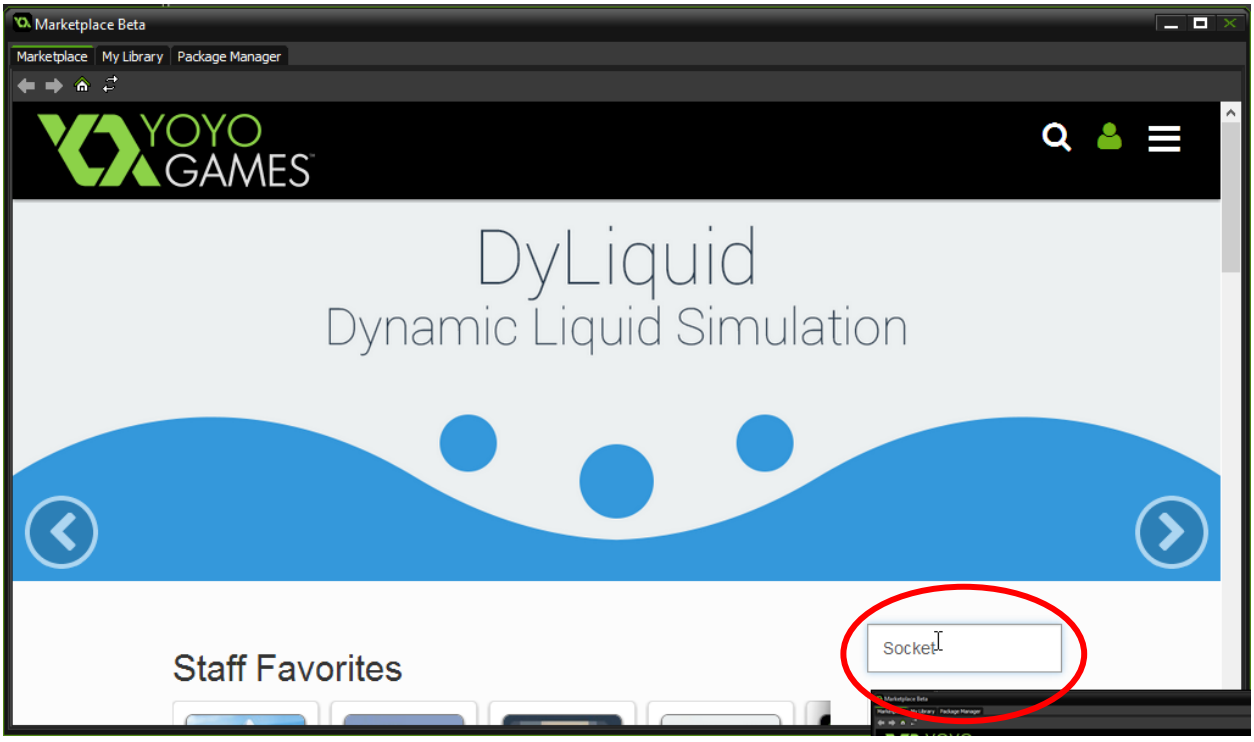
1. Open the GameMaker: Studio project that you want to add the Socket.io extension to.
2. Open the marketplace by clicking Marketplace Beta -> Marketplace.



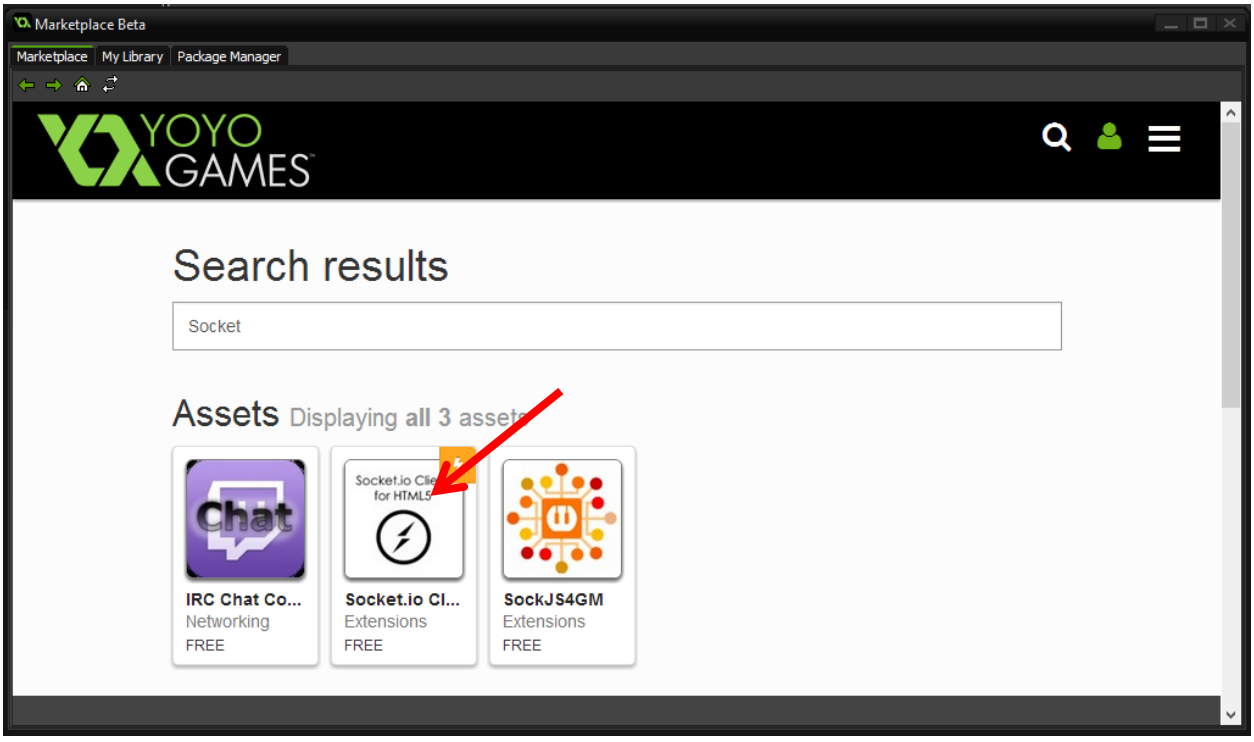
3. Go to the marketplace tab. You should see something like this:



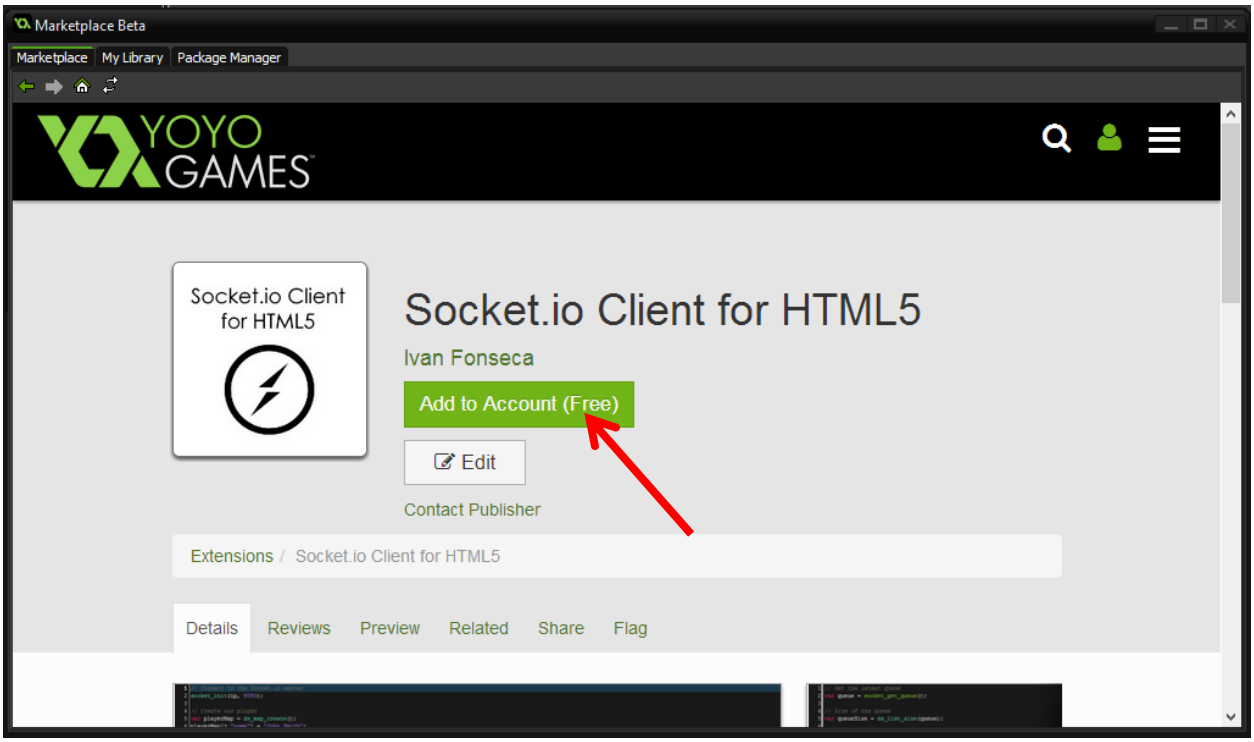
4. In the search box, type “Socket”.



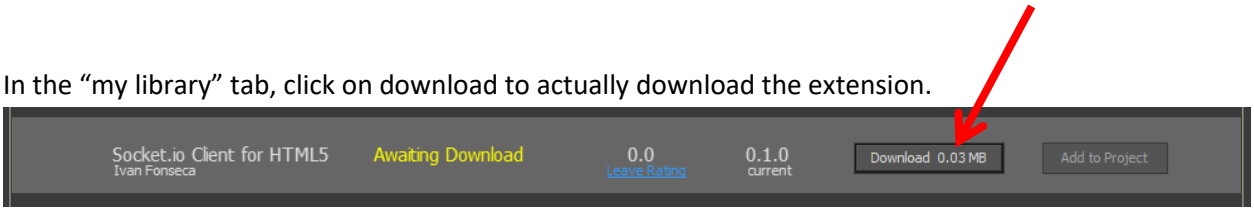
5. The Socket.io Client should be one of the first results. Make sure it has the same name and icon as the picture below:



6. Click on it and you should be brought to the extension page. Press add to account.

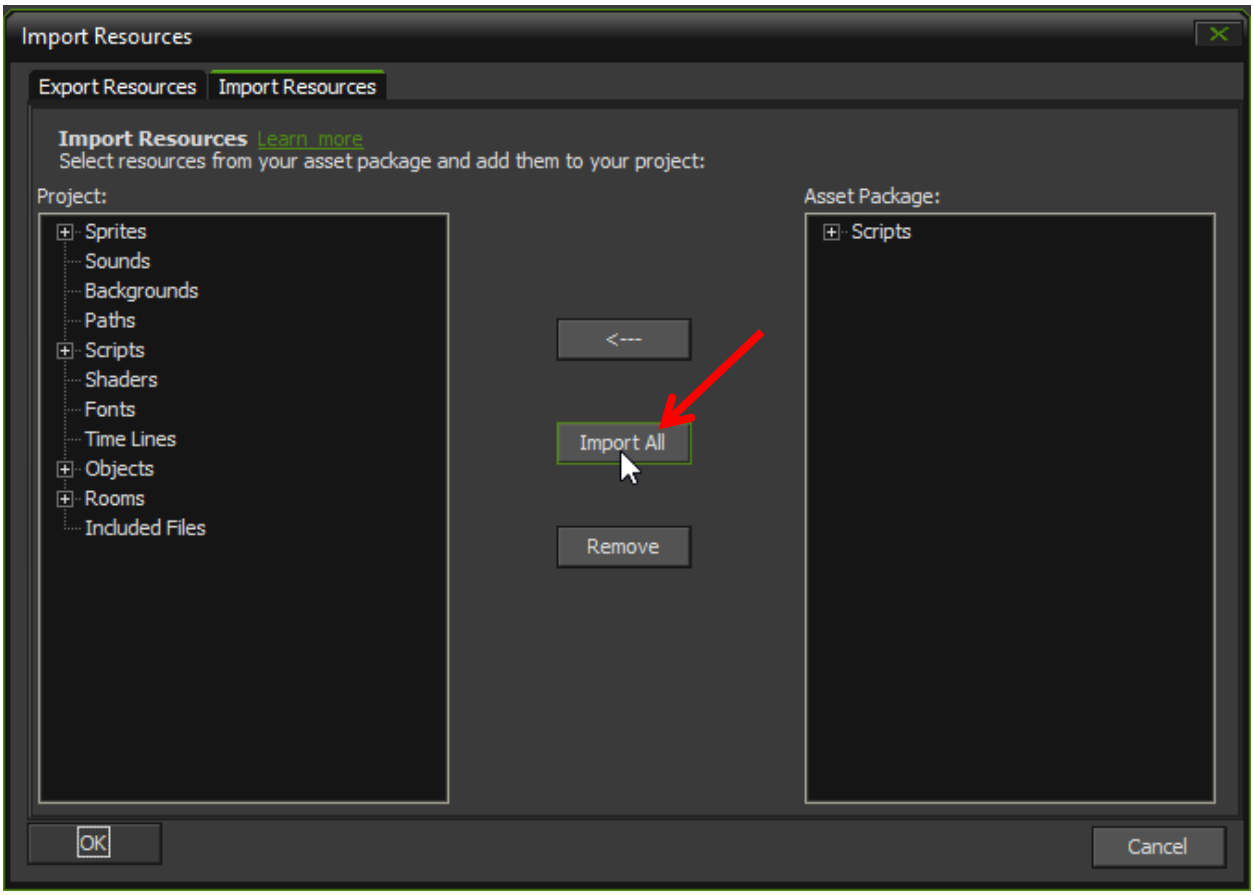


7. In the “my library” tab, click on download to actually download the extension.

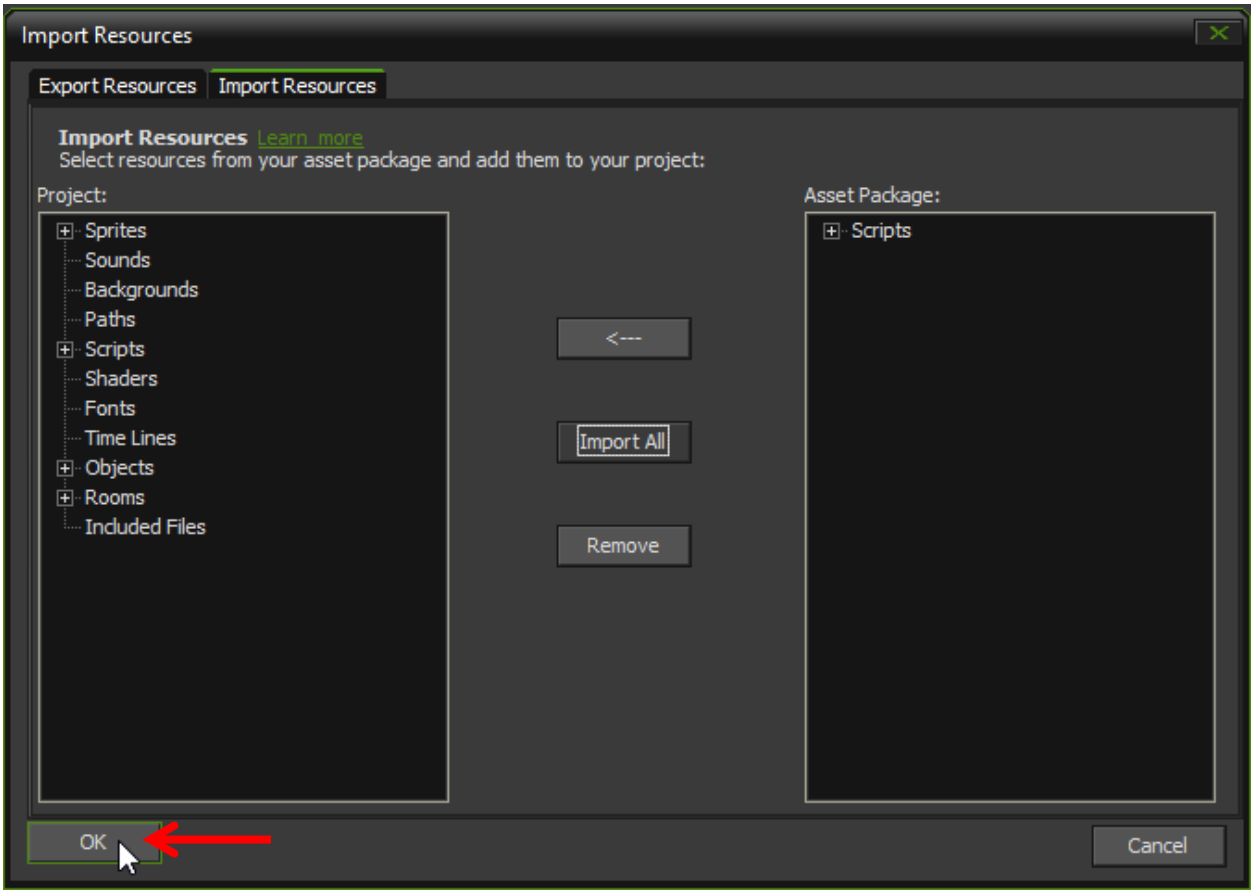


8. Once it finishes downloading, press add to project.

9. In the box that comes up, press import all.



10. Now press OK.



The extension has now been downloaded and added to your project. Note that in the future, you won't have to download the extension again to add it to another project. You can just open the my library tab and it should already be there. Make sure you check the my library tab every once in a while to check for updates.

Tutorial

If you have never used Node.js before, find a simple getting started guide online, You will need some experience with it to write Socket.io servers. The first part of this guide will be about the server. Start a new Node.js project and install Socket.io and Express with `npm install socket.io express -s`. Next, type out or paste the following code into a file called index.js:

```
// Require Socket.io and Express so we can use them in our code
const app = require("express");
const http = require("http").Server(app);
const io = require("socket.io")(http);

// When a new client connects, this will run the code in the squiggly bracket
s
io.on("connection", (socket) => {
    // Write a message in the console so we know someone connected
    console.log("A user connected.");

    // Get greetings from the clients
    // "data" will be a name that the Socket sends us
    socket.on("greeting", (data) => {
        // Send a reply back to the socket
        // "Hello " + data will combine the string "Hello " with the name tha
t the Socket gave us
        // If the socket sent us the name "Dave", we will send them "Hello Da
ve"
        socket.emit("reply", "Hello " + data);
    });
});

// This will make the server start listening for connections
// Here, you need to specify a port. I'm using 3000, but you can use whatever
you want
http.listen(3000, () => {
    // Write a message in the console so we know the server started
    console.log("Server is listening.");
});
```

Reading the comments in the code should give you a good idea of what the code does. Now, how would we use this with GameMaker: Studio? Somewhere in your code (perhaps when the player presses a connect button), you can use `socket_connect()` to connect to a server. `socket_connect()` takes two arguments: the IP address of the server and the port. The IP address should be a string while the port should be a number. If you want to connect to a server running on your computer, you would do `socket_conenct("localhost", 3000)`. Localhost always refers to your current computer. If you changed the port in the server code, replace the 3000 with the port you chose. `socket_connect()` should only be called once (unless you disconnect from the server and want to reconnect later).

Now that we can connect to the server, let's send the server a greeting. To send a message to the server, we use `socket_emit()`. `socket_emit()` takes two arguments: the name of the message you want to send (as you can see in the server code, here it would be called "greeting") and the additional data (in this case, we want to send a name). So, after connecting to the server, somewhere in your code, you want to write `socket_emit("greeting", "Dave")`. This will send a "greeting" message to the server, and out name will be "Dave".

Okay, so now we can send messages, but what if we want to get info back from the server? First, we need to add a listener. A listener will allow us to get a specific type of message from the server. The server can send us whatever type of message it wants, but if we haven't added a listener for that type of message, we'll never get it. If you look in the server code, you'll see that the server responds with a message called "reply". To add a listener, after connecting to the server, write `socket_add_listener("reply", scr_reply).` As you can see, `socket_add_listener()` takes two arguments: the name of the message (a string) and the script that will be called when we get a message of that type (a script index). Here, I've chosen a script called `scr_reply`, but you can pick whatever name you want.

Now, when we get the message of type "reply", the script that we referenced (in my case, scr_reply) will be called. But how do we get the data from the message? When your script called, it will have two arguments: the first one is the name of the message and the second is the extra data (both arguments are strings). So if you want to get the name and data from the message, put this at the top of your script:

```
var name = argument0;
var data = argument1;
```

Now we can use the data from the reply. Since we sent "Dave" as the data in the greeting, the server's response (which is now in the data variable) is "Hello Dave".

If you try running your project, it won't work. We can connect to the server and even send messages to it, but we never get messages back. This is because we aren't checking for updates. In an alarm somewhere, you need to run `socket_update().` This function will take care of messages for you. I don't suggest you run it in a step event as that could cause lag. Depending on the type of game, you may want a fast update rate (a top down shooter or platformer for example), while others can check less often (a card game, for example).

Finally, let's say you want to disconnect from the server. Simple run `socket_disconnect().`

Reference

socket_connect(ip, port);

Connects to a Socket.io server.

Arguments

ip (String): The IP address of the server you want to connect to.
port (Number): The port on the server that you want to connect to.

Example

```
socket_init("123.456.7.8", 1337);
```

---

socket_emit(name, data);

Sends a message to the server.

Arguments

name (String): The type of message you want to send.
data (String): Additional data to send with the message.

Example

```
// Sending simple data
socket_emit(“greeting”, “Hello server!”);

// You can send complex data with a map encoded to a JSON string
// Create a map
var position = ds_map_create();
// Add our x and y position
position[? "x"] = x;
position[? "y"] = y;
// Send it to the server as a JSON string
socket_emit(“position”, json_encode(position));
// Destroy the map when we’re done with it to prevent memory leaks
ds_map_destroy(position);
```

---

socket_add_listener(name, script);

Adds a listener for a type of message from the server.

Arguments

name (String): The type of message you want to listen for.

script (Script Index): The index of the script that's going to handle the message. When you receive a message of this type, the script you reference here will be called with two arguments. `argument0` will be the name of the message. `argument1` will be the extra data.

Example

```
// After connecting, we add a listener
// When we get the message of type "health", our script "scr_get_health" will be called.
socket_add_listener("health", scr_get_health);


// In scr_get_health, we can get the type of message
// and data from argument0 and argument1
var name = argument0;
var data = argument1;
```

---

socket_disconnect();

Disconnect from the Socket.io server we're currently connected to.

Example

```
// This will disconnect us from the Socket.io server
socket_disconnect();
```