WSChat WebSocket Protocol for the SocialVP

Aaron Olkin

July 8, 2013

Background on WebSockets

The SVP needs a framework for providing enhanced chat functionality, and since WebSockets are so cool, new, and useful, it uses them. WebSockets provide asynchronous full duplex communication between a web browser and a server, allowing either side to send events without anything from the other end. This means that clients can recieve messages sent by other clients without any form of server polling, just one long-running connection.

1 Introduction

The SVP Project uses the websocket protocol to implement an enhanced chat room interface for watching a video together across the internet. It works by exchanging JSON objects with the server, which then distributes them to the appropriate client. The server actually has very little part, since the WebSocket server performs very little management beyond connection housekeeping. In this way, most of the functionality is client-side javascript exchanging whatever messages it wants to.

As a convention in this manual, monospace text refers to literal dictionary key or value names. Further, double quotes (") refers to a dictionary key, while a single quoted (‘) string is a literal value.

2 Usage

Everything sent to the server should be a JSON encoded dictionary with a `"command"` field and possibly others depending on the command. Responses from the server will always at least have a `"type"`field.

2.1 Establishing a Connection

Upon creation of a websocket pointing to the websocket server, your client should recieve a message of `"type"` `‘init’` with a welcome `"message"`. You may safely ignore this message, as long as you follow the rest of the protocol.

In order to send and recieve further, you must identify yourself with a nickname. Any non-identification commands sent before identifying will result in you recieving a dictionary of `"type"` `‘outcome’`with an error message warning you to identify. To identify, send an identification command. The `"command"` field value here should be `‘identify’`, and there should be another field, `"name"`, specifying the name you wish your client to use and be contacted by. After sending this, you should recieve a message of `"type"` `‘outcome’`either with `"success"` set to `True` or `False` depending on whether that name was successfully claimed for you. Either way, you will also get an informational `"message"`.

2.2 Sending Messages

Once a connection is established and you have identified yourself, you may begin sending messages. The server only acts as a broker between several clients. It does not care what data is being exchanged at all, so clients are free to send and recieve whatever "messages" they want, and the other clients must deal with parsing whatever data is sent their way.

The server accepts two `"command"`s for sending messages, `'broadcast'` and `'message'`. `'broadcast'` sends the `"message"`to all other identified clients who are connected to the server. `'message'` sends the `"message"`to the client specified in the `"to"` field of the request. If no client with that name is connected to the server, it send a message of `"type"` `'outcome'`with `"success"` set to `False` and a descriptive error `"message"`.

2.3 Recieving Messages

When another connected client sends a `'broadcast'`, all other clients recieve a message of `"type"` `'broadcast'` with a `"from"` field set to the name of the client who sent the broadcast. They also recieve the `"message"`payload sent in the broadcast in a `"message"`field.

When a client sends a message of `"type"` `'message'`, the client whose name is specified in the message's `"to"` field will recieve a message of `"type"` `'message'`. This message will also have a `"from"` field containing the sender's name, as well as a `"message"`field containing the sent message.

Messages of `"type"` `'outcome'`should only be recieved in response to commands sent by the client, telling whether the requested action was successful or not, and why.

3 `'outcome'` Message "Codes"

10 ***"Please identify yourself"***
: This code is sent when a client tries to use a command before identifying with a name.

300 ***"Indentified successfully!"***
: This code is sent with a `True` `"success"` field, indicating that the name used to identify is unique and was accepted.

305 ***"Client with that name already exists!"***
This code is sent after attempting to identify when the client has chosen a name that is already taken. A client recieving this code should `identify` again using a different name.

400 ***"Missing required request field!"***
This code is sent when a client sends a `message` to the server, but is missing the `"command"` field or a field required by that command.

404 ***"No client with that name is currently connected!"***
This code is sent after a client tries to `message` another, but the client they tried to message does not exist or is not currently connected to the server.