IRC Chat Protocol Specifications
Carter Wilson

1. Introduction

The following document describes an Internet Relay Chat protocol by which clients can communicate with each other in real-time via a single server. Users can join chat rooms and broadcast messages out to all other clients in the room, or send messages directly to other clients.

2. Conventions used in this document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, AND “OPTIONAL” in this document are to be interpreted as described in RFC 2119. In this document, theses words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

3. Basic Information

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on port 1025. Clients connect to this port and maintain this persistent connection to the server. The client can send messages and requests to the server over this open channel, and the server can reply via the same. This messaging protocol is asynchronous in that the client is free to send messages to the server at any time, and the server may asynchronously send messages back to the client. Both the server and client can terminate the connection at any time without degrading or halting the performance of the other.

4. Message Infrastructure

4.1. Generic Message Format

*messages are sent in json format

```
Message = {
      “command” : <string>,
      “nickname” : <string>,
      “room” : <string>,
      “message”: <string>
}
```

4.1.1. Field Definitions

4.1.1.1. command - specifies what type of message is being sent.

4.1.1.2. nickname - specifies the nickname of the client sending or receiving the message.

4.1.1.3. room - specifies the chat “room” that the message is intended for, or in the case of a direct message, the recipient nickname.

4.1.1.4. message - contains the actual text of the message that should be displayed.

4.1.2. Command Codes

4.1.2.1. /nick

4.1.2.2. /join

4.1.2.3. /broadcast

4.1.2.4. /direct

4.1.2.5. /listRooms

4.1.2.6. /leaveRoom

4.1.2.7. /roomMembers

4.1.2.8. /close

4.1.2.9. /joinSuccess

4.1.2.10. /nicknameSet

5. Error Messages

Due to server design, only one error message should be necessary, that for failing to set a nickname correctly. All other operational errors will be captured in the message text sent by the server.

5.1. Failed to Set Nickname

```
message = {
      “command” : “/nickFail”,
      “nickname” : <chosen nickname>,
      “room” : “”,
      “message” : <msgText>
}
```

6. Client Messages

6.1. First Message sent to the server

```
message = {
```

```
"command" : "/nick",
"nickname" : <chosen nickname>,
"room" : "",
"message" : ""
}
```

6.1.1. Usage

Before subsequent messages can be sent, a connecting client MUST provide a nickname. The server MUST associate the client's nickname with the socket connection of the user. This message SHOULD only be sent once; if the server receives it more than once it can ignore it.

6.1.2. Field Definitions

- 6.1.2.1. "command" - the command for the message, MUST be "/nick" for first message.
- 6.1.2.2. nickname - nickname the client has chosen to associate with their socket connection on the server.
- 6.1.2.3. nickname has to be unique, otherwise the server MUST return an error informing the client that it is taken.
- 6.1.2.4. room - not applicable for this message, leave blank.
- 6.1.2.5. message - not applicable for this message, leave blank.

6.2. Listing Rooms

```
Message = {
"command" : "/listRooms",
"nickname" : <client nickname>,
"room" : "",
"message" : ""
}
```

6.2.1. Usage

Sent by the client to request a list of all rooms currently stored on the server.

6.2.2. Response

Server MUST return a response in which response[“command”] = “/joinSuccess” and response[“message”] contains the list of all chat rooms.

6.3. Joining and Creating Rooms

```
message = {
      “command” : “/join”,
      “nickname” : <client nickname>,
      “room” : <room name to create or join>,
      “message” : “”
}
```

6.3.1. Usage

Sent by the client to join a chat room. If no room by that name exists, one is created.

Upon joining a room, the server MUST send a response to the client in which response[“command”] = “/joinSuccess”. The server MUST update the list of clients in the room to include the client socket, and the client MUST update its list of rooms it is in upon receipt of the response.

6.4. Leaving a Room

```
message = {
      “command” : “/leaveRoom”,
      “nickname” : <client nickname>,
      “room” : <room to leave>,
      “message” : “”
}
```

6.4.1. Usage

Sent by the client to leave a chat room.

Upon receiving this message the server MUST remove the client form the specified room. The server SHOULD send a json message with message[“command”] = “/leaveRoom” and the message text in message[“message”] informing all other clients in the

room that the user has left.

6.5. Broadcasting Messages

```
message = {
      “command” : “/broadcast”,
      “nickname” : <client nickname>,
      “room” : <target room>,
      “message” : <msgText>
}
```

6.5.1. Usage

Sent by a client to send a text message to every other client in a room.

Upon receiving the message the server MUST send the same message to all other clients in the specified room. It is left to the client to format the message in whatever way makes sense using the json provided.

6.5.2. Field Definitions

Target room - room to which the message is being sent to

msgText - text meant to be sent to other clients in the room. The text MUST consist entirely of readable ASCII character values

6.6. Sending Direct Messages

```
message = {
      “command” : “/direct”,
      “nickname” : <client nickname>,
      “room” : <target nickname>,
      “message” : <msgText>
}
```

6.6.1. Usage

Sent by client to send a message to another specific client on the server.

Upon receipt of the message, the server MUST forward

to the message to the specified client.

6.6.2. Field Definitions

Target nickname - nickname of the client on the server that message is being sent to.

msgText follows the same rules as the text for a broadcast message.

6.7. List Members of a Room

```
Message = {
     “Command” : “/roomMembers”,
     “nickname” : <client nickname>,
     “room” : ‘’,
     “message” : ‘’
}
```

6.7.1. Usage

Sent by a client to find out which users are subscribed to the specified room.

The server MUST respond by sending back a message to the requesting client with a listing of all the members currently subscribed to the specified room.

6.8. Close Connection with Server

```
Message = {
     “Command” : “/close”,
     “nickname” : <client nickname>,
     “room” : ‘’,
     “message” : ‘’
}
```

6.8.1. Usage

Sent by the client with the intent to close their socket connection with the server. Upon receipt of the message, the server MUST close the socket connection.

7. Server Messages

7.1. List Response

```
Message = {
      “Command” : “/listRooms” or “listMembers”,
      “nickname” : <client nickname>,
      “room” : ‘’(for listRooms) or <room>(for
listMembers),
      “message” : <list>
}
```

7.1.1. Usage

Sent by the server in response to a client request to retrieve a list of either all rooms or all members in a certain room. It is important to note, the list is not a programmatic list; it is a string built by the server.

7.2. Successful Room Join Response

```
Message = {
      “Command” : “/joinSuccess”,
      “nickname” : <client nickname>,
      “room” : <room joined>,
      “message” : <msgText>
}
```

7.2.1. Usage

This message will be sent to a client from the server upon a successful subscription of a client to a given room. It includes a short message describing which room was joined.

7.3. Nickname Set Successfully

```
Message = {
      “Command” : “/nicknameSet”,
      “nickname” : <client nickname>,
      “room” : ‘’,
      “message” : <msgText>
```

```
}
```

7.3.1. Usage

This message will be sent to a client from the server after the server has successfully fulfilled a request from the client to set its nickname. It includes a short message confirming the user's nickname.

8. Error Handling

The server and client both MUST be able to detect when the socket connection linking the two is broken. If the connection is broken, the client MUST notify the user, and the server MUST also print or log a notification. The server MUST remove the socket from all incoming and outgoing lists, remove the socket from its dictionary of sockets, and remove any outgoing messages for that socket from the message queues for the sockets. If the client is disconnected, it must be restarted to reconnect to the server.

9. Extra Credit Features Supported

- Private messaging is supported as described above
- A cloud-connected server is supported. The server is hosted on Google Cloud in a virtual machine, and due to cost constraints, must be started manually for the purpose of demoing the project.

10. Security Considerations

This application provides no protection against any kind of tampering, interception, or viewing of messaging by outside parties. Private messaging goes directly to a specified user, but is no more protected from outside parties than normal broadcast messaging. It is recommended that users either implement their own security or do not send sensitive data using the application.