CS 494 L. Voepel
Internet Draft Portland State University
Intended status: IRC Project Specification June 9, 2019
Expires: December 2019

IRC Chat Client
draft-irc-pdx-cs494-00.txt

Status of this Memo

Copyright Notice

Abstract

The following memo describes an IRC like Server/Client chat protocol developed for CS 494: Internetworking Protocols at Portland State University. The Protocol is intended for use by a single server handling multiple clients communicating between one another.

Table of Contents

1. Introduction

This is a specification for an internet relay chat (IRC) protocol. The system consists of a single server which will relay information between multiple connected clients.

Functions for users include: joining/creating a room, asking for a list of existing rooms, listing users in a given list of rooms, and messaging users in a given list of rooms. Users can also private message another user outside of a room.

Rooms consist of a list of users which are “within” them and a room name.

Users consist of a username, a list of rooms they are connected to, and socket/address information for relaying messages.

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 [RFC2119].

In this document, these 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.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. Basic information

Communication described in this protocol takes plas over TCP/IP with the server listening for connections on port 1234. Clients connect to this port and communications between the server and client are relayed using this connection.

4. Client Messages

4.1. Message Format for Client to Server

Messages are relayed from the client to the server in the form of a dictionary with 3 key/value pairs:

```
Dict_to_send = {'cmd': command,
                 'msg': message,
                 'list': recipientList}
```

4.1.1. Field Definitions:

- command – specifies what client wants the server to do with the data sent. Formatted as a string

- message – contains data to be relayed. Formatted as a string

- recipientList – specifies list of users/rooms for message to be relayed to. Formatted specifically as a list datatype

4.1.2. Commands (cmd):

Reg, Join, Msg, RList, UList, Exit, PM

4.2. Registering with Server

```
Dict_to_send = {'cmd': “Reg”,
                'msg': username,
                'list': “”}
```

4.2.1. Usage

Before being able to send messages to other clients, the client must register using an unused username. The server MUST associate this name with the client only if the username isn’t already in the database. Otherwise, it will notify the user that the name is in use and register it without a name. It will prevent the user from communicating until a name is filled.

4.3. Joining Rooms

```
Dict_to_send = {'cmd': “Join”,
                'msg': “”,
                'list': RoomsToJoin}
```

4.3.1. Usage

Sent by the client to join a room. If room name is not in use, creates a new room.

Upon registering a user with a room, the server MUST notify all room members of the user’s entry.

Users MUST be notified by server when list of users in a room changes.

4.4. Messaging a room

```
Dict_to_send = {'cmd': “Msg”,
                'msg': MessageToSend,
                'list': RoomsToJoin}
```

4.4.1. Usage

Sent by client to message a given list of chat rooms.

If client is a member of a given room within the list, the server MUST relay the message to all users in the given room with the exception of the sending user. Server MUST also provide the name of the client the message was sent from and the name of the room the client was messaging.

4.5. Listing Rooms

```
Dict_to_send = {'cmd': “RList”,
                'msg': “”,
                'list': “”}
```

4.5.1. Usage

Sent by client to receive list of rooms on the server.

Server MUST respond with a list of all rooms on the server.

4.6. Listing Users in given rooms

```
Dict_to_send = {'cmd': “UList”,
                'msg': “”,
                'list': ListOfRoomsToSee}
```

4.6.1. Usage

Sent by client to receive list of users in the given rooms.

For each room in list, server MUST respond with a list of all usernames for users connected to the room.

4.7. Leaving Rooms

```
Dict_to_send = {'cmd': “Exit”,
                   'msg': “”,
                   'list': ListOfRoomsToLeave}
```

4.7.1. Usage

Sent by the client to leave a list of rooms.

For each room in list, server MUST remove room from client’s list of rooms, remove client from room’s list of clients, and notify all members of the given room of client’s departure.

4.8. Private Messaging

```
Dict_to_send = {'cmd': “PM”,
                'msg': message,
                'list': ListOfUsersToMessage}
```

4.8.1. Usage

Sent by client to relay a private message directly to a list of other clients.

For each client in the list, server MUST send the private message along with the user sending the message

5. Server Messages

5.1. Message Format for Server to Client

Messages are relayed from the server to the client in the form of a dictionary with 4 key/value pairs:

```
package = {"type": messageType,
           "room": sendingRoom,
           "user": sendingUser,
           "msg": message}
```

5.1.1. Field Definitions:

o messageType - specifies what type of message is being sent

o sendingRoom - specifies what room message was sent to (if any)

- sendingUser – specifies what user message was sent from (if any)

- message – message to be relayed to client

5.2. Relaying Room Message

```
package = {"type": “room_msg”,
           "room": sendingRoom,
           "user": sendingUser,
           "msg": message}
```

5.2.1. Usage

Sent by server to relay message sent by client to a given room.

5.3. Relaying Private Message

```
package = {"type": “PM”,
           "room": “”,
           "user": sendingUser,
           "msg": message}
```

5.3.1. Usage

Sent by server to relay message form client directly to another client.

5.4. Notify

```
package = {"type": “notify”,
           "room": sendingRoom,
           "user": “”,
           "msg": message}
```

5.4.1. Usage

Sent by server to notify a client of a change on the server which directly affects the given user. Server MUST include room if this message is to notify users of another user leaving or entering a room.

5.5. Error

```
package = {"type": “room_msg”,
           "room": sendingRoom,
           "user": sendingUser,
           "msg": message}
```

5.5.1. Usage

Sent by the server to a client to notify client of a non-critical error. Server MUST send when client is attempting to connect with already registered username.

6. Error Handling

Server and client MUST both be able to detect when connection between them is terminated. If either party detect the connection as being lost, then they MUST consider the other to be disconnected. If the server detects a client disconnect, it MUST notify all rooms the client was in of the client leaving, and MUST also close the socket the client was connected to. If the client detects that the server has been lost, the client MAY try to reconnect.

7. Extra Features

Note that private messaging was an extra feature which went beyond the minimum project requirements.

8. Security Considerations

Messages sent over this system have no protection against tampering and are not encrypted. The term “Private Messaging” only refers to the fact that the messages will not be sent to other clients on a by room basis.

9. IANA Considerations

None

10. Conclusions

This document is meant to outline a framework for an IRC protocol. Using this document as a guide, clients can devise their own protocols.

11. References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
Informative References

11.1. Normative References

[2] Oikarinen, J., "Internet Relay Chat Protocol", RFC 1459, May 1993.

[RFC1459]Oikarinen, J., "Internet Relay Chat Protocol", RFC 1459, May 1993.

Acknowledgments

This paper would like to acknowledge Portland State University and the instructor of CS 494, Nirupama Bulusu

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Lauren Voepel
Student

Email: lvoepel@pdx.edu