Working Group Name Shweta Korulkar

Deepa Varghese

Internet Draft Portland State University
Intended status: IRC Class Project Specification October 30, 2018

Expires: Nov 2018

Internet Relay Chat Class Project
draft-irc-pdx-cs594-00.txt

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to publish it as an RFC and to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html

This Internet-Draft will expire on December 31, 2018.

Copyright Notice

Abstract

This memo describes the communication protocol for an IRC-style client/server system for the Internetworking Protocols class at Portland State University.

Table of Contents

1. Introduction.....2
2. Conventions used in this document.....3
3. Basic Information.....3
4. Message Infrastructure.....3
4.1. Generic Message Format 3
4.2. Error Message..... 3
5. Client Messages..... 3
5.1 First message sent to server.....….....3
5.1.1. Usage 4
5.2 Listing Room..... 4
5.3 Creating Room..... 4
5.4 Joining Room..... 4
5.5 Leaving a Room..... 5
5.6 List online clients..... 5
5.7 Main Menu. 5
5.8 Personal Message..... 5
5.9 Room Message. 6
6. Server Messages. 6
6.1. Listing Responses..... 6
6.2. Forwarding Messages. 6
7. Error Handling.... 6
8. “Extra” Features Supported.....….....6
8.1. Private Messaging.....…..... 6
8.2. Broadcast.....…..... 7
9. Conclusion 7
10. Normative References.....……..... 7
11. Acknowledgements... .....……..... 7

1.Introduction

This specification describes an Internet Relay Chat (IRC) protocol by which clients can communicate with each other via a central Server, that ‘relays’ messages that are sent to other connected users.

Users can create new “rooms” and add some/all existing online clients, such that any message sent by any user subscribed to a room is broadcast to all other clients in that room, they can join an existing room and leave any room. Users can also send

private messages directly to other users. User can quit the program, thus getting disconnected with server which will also remove the said user from all the groups that the user had joined.

2.Conventions used in this document

Values within the code piece presented in this document, and in the text of this document that are dynamically allocated or hard-coded will be enclosed within ##

3.Basic Information

The Application rests on the transport layer protocol of TCP, in a client-server architecture. The server is “always on”, and the client connects to the server by sending a “connection request”. The server responds with an appropriate “response message”. This is the application level handshaking after which a persistent connection is established, and the client can communicate with the server, asynchronously.
Error handling code is available at both ends, thus a client will be gracefully informed if the server is unavailable or facing high traffic. The server will also handle client-side issues gracefully.

4. Message Infrastructure

4.1 Generic Message Format

To enter command:
    **#command#
To send private message to person or group:
    #message#**#person/group user-name
Message received from person or group:
    >>#user/group#: #message#
Message received from server:
    #server’s message#

4.2. Error Message

If the text entered does not follow the above format, then the server identifies it as invalid message and returns with the request that the client enter a corrected command or new command

5. Client Messages

5.1 First message sent to server

The client is prompted to enter a username. This username is sent to the server. The server validates if the username is unique. If not, then the client is requested to enter a new name. This repeats till a unique name is entered, or the client quits the application.

5.1.1. Usage

Once the connection has been established, and before any other communication can occur, the client must provide a unique username.

The server associates this name with the client’s socket connection, using a hashmap whose key is the name of the client and value is the socket connection of that client.

5.2 Listing Room

If user want to see the list of existing groups then user should enter **list_room

Command is used by client to find the existing rooms.

The server returns with a list of existing rooms and its currently active members. It maintains a hashmap whose key is name of the room, and value is the list of clients registered to that room.

5.3 Creating Room

To create a room: **create_room #name_of_room#

Command is sent by the client to create a chat room. If no room by that name exists, one is created.

The server registers the new room, with the client added by default.

5.4 Joining Room

To join the room: **join_room #name_of_room#

Command is sent by the client to join any existing room.

The server responds with the list of existing rooms and their members.

5.5 Leaving a Room

To leave a room: **leave_room #name_of_room

Sent by the client to leave a chat room.

The server removes the client’s registration from that particular room.

5.6 List online clients

If user wants to see which clients are online then user should enter **online_clients

Sent by the client to see the user who are currently online.

Server returns the keys of the hashmap clients where it has maintained the list of client socket connections currently alive.

5.7 Main Menu

To see all commands user should type **help.

The client code displays the list of commands available to the user.

5.8 Personal Message

To send personal messages, user should enter

#some message# **#name of the other client#

Sent by the client to some other online client, visible only to that client.

Server receives the message, parses the string to obtain the name of the other client, and sends the message on the other client’s socket connection.

5.9 Room Message

To send a message to a Room, user should enter

#some message# **#room’s name

Sent by client to all the members logged in to the room

Server iterates over the value of the hashmap where key is the name of the group, and sends a message to all connected clients.

6. Server Messages

6.1. Listing Responses

Two types of hashmaps maintained by the server: one is to record all the online clients, two is to record existing group names and their registered clients. Hashmap values are converted to string and sent.

send(#message#.encode())

6.2. Forwarding Messages

Messages received from a client is parsed to investigate the purpose of the message. If the message is meant for another client, the message is passed to that client via it’s socket connection.

If the message is meant to be sent to a room, the server iterates over the list of members in the room and forwards the message to each client via their individual socket connection.

send(#message#.encode())

7. Error Handling

If the server detects that the client connection has been lost, the server MUST remove the client from all rooms to which they are joined. If the server disconnects all the client receives a prompt that the server has been disconnected and they should shut the connection. They may choose to connect to the server again, if the server is up.

8. “Extra” Features Supported

8.1. Private Messaging

Note that private messaging is supported in addition to meeting the other remaining project criteria.

8.2. Broadcast

The client can send a common message to all online clients. The format of the message is similar to the existing format. For broadcast it is: #message# **broadcast

9. Conclusion

This specification provides a generic message passing framework for multiple clients to communicate with each other via a central forwarding server, employing the Transport Layer Protocol.

This specification is the first draft of the IRC Project henceforth undertaken, which considers the transport layer protocols and the services provided by it. Based on which a plan of the application has been created, a framework through which multiple clients can communicate with each other via a server.

10. Normative 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

11. Acknowledgements

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