**CS594**

**Internet Draft**

**Intended Status: IRC Class Project Specification**

GROUP MEMBERS: HANESH KOGANTI

POLEPALLI SAI GANESH

MOUNISHA KONDURU

# Internet Relay Chat

## Status of this Memo

This Internet-Draft is presented in complete accordance with BCP 78 and 79's guidelines. This document may only be altered to publish it as an RFC and to translate it into languages other than English. No derivative works of this document may be generated.

The Internet Engineering Task Force (IETF), its sections, and its working groups produce Internet-Drafts as working documents. The distribution of working materials as Internet Drafts by other organizations should be noted.

http://www.ietf.org/ietf/1id-abstracts.txt provides access to the list of active Internet-Drafts.

http://www.ietf.org/shadow.html to get the list of Internet-Draft Shadow Directories.

On Fail 1, 0000, this Internet-Draft will become invalid.

**ABSTRACT**

For the Internetworking Protocols course at Portland State University, this letter defines the communication protocol for an IRC-style client/server system.

## Tables of Contents

**1. Introduction** 4

**2. Conventions used in this document** **4**

**3. Basic Information** **5**

**4. Message Infrastructure** 6

4.1. Generic Message Format 6

4.2. Error Messages 6

**5. Client Message** 7

5.1. First message sent to server 7

5.1.1 Usage 7

5.2. Creating Room 7

5.3. Joining Room 7

5.4. Listing Room 8

5.5. Listing members in a room 8

5.6. List online clients 8

5.7. Leaving Room 8

5.8. Private chat 8

5.9. Group chat 9

**6. Server Messages** 9

6.1. Listing Responses 9

6.2. Forwarding Messages 9

**7. Error Handling** **10**

**8. Miscellaneous Features** 11

8.1. Broadcast Messaging 11

**9. Conclusion and Security consideration 11**

**10. References 11**

**11. Acknowledge** 12

## 1. Introduction

This RFC specifies a protocol that enables the clients to connect on a shared server for inter client communication or chat using the Internet Working Protocol known as TCP/IP. Here we designed an Internet Relay Chat (IRC) over TCP/IP protocol as a foundation and it serves as a function for clients to communicate with each other. This system operates on a client server architecture and the server acts as a main hub to transfer messages sent to it to all other users connected to it based on the protocols.

A Client is a User who can establish their own rooms and join them here, and the client can also broadcast their message to the other clients in any room, or they can join any room that is created by other clients and is already connected to the server. Here there is also a listing function which will be helpful for listing all the rooms that are available, and a client can join any preferred space and can send a message in any room as needed. Finally, a client can also send private messages to the other users directly.

## 2. Conventions used in the document

Commands that are dynamically allocated and are used within the code fragment are described within this article, as a text of this document. All these Commands are considered as keywords and will be wrapped by []. This convention helps reviewers to locate the sections of this RFC that are covered by these keywords fast.

**List of commands or Keywords:**

- Create or join a room:

  join-room [room-name]

- To send message to a room:

  chat-room [room-name] [message]

- To send private message to person:

  pvt-msg [recipient-name] [message]

- To display the all the users:

  List users

- To display all the members in the room:

List members [room-name]

- To display the rooms

  List rooms

- Broadcast message to all active users:

  Broadcast-msg all [message]

- To exit from a room:

  exit-room [room-name]

- To quit from IRC:

  quit-irc

## 3. Basic Information

The Transmission Control Protocol (TCP), which employs client-server architecture, is the main foundation of this project and all communication in this IRC takes place over this protocol with the server listening for connections on port number 1234. In this case, the server is in "Listening Mode" and is constantly listening to clients. The same open channel is used by different clients to connect to the server, and they will send the messages through a specified port that is specific to that server asynchronously and the server replies via the same channel to that client.

As soon as the client joins the server, The "Enter your UserName: " message is displayed in this program and the user must enter their username to continue chatting with other clients which can done by joining a specific room or by broadcasting. The constraints on the client username include the usage of 10 alphanumeric characters, including an underscore and this name can be used to identify the user.

As a user connects, a message like " You have enrolled successfully" is displayed to the client. After connecting to the server, a user can join a room by creating it if it doesn’t exist or can join an existing room by using Join-room command.

Here we also included features such as sending a private message to the users who ever connected to server and a client can even also leave a particular room and disconnect from the server the moment he wanted.

We have also included Error handling code on both the server and the client slide. The client will be informed if the server is down or if it is experiencing any other problems and the server will take care of the client-side issue.

# 4. Message Infrastructure

## 4.1. Generic Message Format

**To send a message a client can use below specified command format :**

- To send message to a room:

  chat-room [room-name] [message]

- To send private message to person:

  pvt-msg [recipient-name] [message]

- Broadcast message to all active users:

  Broadcast-msg all [message]

## 4.2. Error message

If the commands are not entered in the above-mentioned format, then both the server or the client will throw an error message and will also display the correct format of the command to be entered to the user.

# 5. Client Message

## 5.1 First message sent to server

The user is prompted by the client to enter their username. The client transmits the server the user's entered username. To ensure that the username provided by the user is distinct, the server performs validation on it. The server asks the user to input the password if the username they entered is distinct. The client-server connection is then created once the user enters the password. The server will ask the user to enter the proper username if the username they entered is not unique. After several cycles, the server terminates the program. This procedure continues until the user gives valid credentials.

### 5.1.1. Usage

The client must provide a distinct username after the connection has been made for any further communication to proceed. By utilizing a HashMap with the client's name as the key and the client's socket connection as the value, the server can link this name to the client's socket connection.

## 5.2 Creating Room

To create a room: join-room [room-name]

The user gives the order to start a chat room. If there isn't already a room with that name, one is created, and the server registers the new room, which by default includes the client. The same command is used to create and join the room. If the requested room is not already present, one is created, and the user is added to it.

## 5.3 Joining Room

To join the room: join-room [room-name]

To join any already-existing room, the client issues a command to the server. There are three scenarios that might occur: First, the server invites the user to the room if it already exists and he or she is not a member of it. Two, the server delivers an error

message if the user is already in the room. If the specified room doesn't already exist, the server will build one, and the user will be added to it.

### 5.4 Listing Room

To list rooms: list rooms

The user must input the above instruction for every room that the client lists as being available.

### 5.5 Listing members in a room

To list members in a room: list members [room-name]

The client sends a request to obtain a list of every user in a particular room. A list of the current users in that room is then returned by the server. This list is maintained by the server using HashMap.

### 5.6 List online clients

To list online clients: list users

To view a list of all currently active clients, the client issues a command. The server then returns a list of all users who are currently logged in. A HashMap is used by the server to maintain this list.

### 5.7 Leaving Room

To leave a room: exit-room [room-name]

The customer commands to leave the chat room. In response to the message, the server either removes the client's registration from that room if the user is a member of that room or provides an error message if they are not.

### 5.8 Private Message

To send Private messages to other users: pvt-msg [recipient-name] [message].

The user can send a private message to a particular user by using the above command where it will reach only that particular user if he has connected to the server.

### 5.9 Group chat

To send a message to a room/group:chat-room [room-name] [message]

All logged-in members of the room receive the message from the client. The server delivers a message to every client that is currently connected after iterating through the HashMap's value, where the key is the group name.

## 6. Server Messages

### 6.1 Listing Responses

**client.send(.encode())**

Two separate hashmap types are stored on the server. The first is to monitor online users, and the second is to track already-used room names and their logged-in users. Prior to transfer, the hashmap values are converted to strings.

### 6.2 Forwarding Messages

**clientusers[i].send(.encode())**

Parsing communications that come in from clients allows us to look into their intended use. If another client is the intended recipient of the message, it is forwarded to that client via its socket connection. If a room is the intended recipient of the message, the server transmits the message to each client over a separate socket connection after iterating through the list of the room's members.

## 7.Error Handling

Faults are available in two forms one when the client or server disconnects, and the other in between. The moment a client loses connectivity, the server recognizes it and kicks them out of all rooms they have joined. All clients are immediately informed when a server disconnects that the connection has been closed and the server has been terminated. The clients have the option to reconnect once the server is running.

Username_Error1: To enter username without any spaces

Username_Error2: The username cannot be empty

Username_Error3: Maximum length allowed for username is 10

Username_Error4: Maximum number of characters allowed for room name is 10

Argument_Error1: Command have a less number of arguments then intended

Argument_Error2: Command have more no of arguments then intended

Argument_Error3: Invalid arguments entered

Message_Error1: Unauthorized message\n You do not have the Authorization for sending this message

Message_Error2: The message size is too large

Command_Error1: Invalid Command

Command_Error2: Command executed more than once for a given set of arguments

IRC_Registration_Error: Username is already in use

## 8. Miscellaneous Features

### 8.1 Broadcast Messaging

Broadcast is to share messages to multiple clients. The command description and explanation are as follows:

To broadcast a message, we use: broadcast all [message]

The client sends a command directing the server to send the same message in the same format to all online clients.

## 9. Conclusion and Security consideration

With the help of a centralized forwarding server and TCP, this standard offers a general message carrying architecture for numerous clients to interact. Clients may design their own protocols that depend on the text-passing system as specified here without making any changes to this specification. The Internet Relay Chat (IRC) Project's initial specification is presented in this document which is a framework that allows numerous clients to connect with one another via a server and has been developed based on the plan of the application. The messages can only be accessed by the users at the endpoints i.e., at client side or server side if it is a private message or can be accessed only by the users in the room if the message is sent by the client in the specified room. In future we are also thinking of adding encryption message techniques and file transfer mechanisms to the existing IRC.

## 10. References

Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

https://datatracker.ietf.org/doc/html/rfc2119

https://en.wikipedia.org/wiki/Internet_Relay_Chat

## 11. Acknowledge

For these project deliverables, we used Google resources and the Google product suite. For the design, we also researched Wikipedia and other sources.