Rajat Kulkarni
Marten Sova
Timofey Shichalin

6/1/22

Internet Relay Chat Project RFC

Abstract

This document is an outline of the Internet Relay Chat application and its communication protocol.

Status of This Memo

This is an Internet-Draft document (work in progress) and is not intended for publication.

Copyright Notice

Table of Contents

1. Introduction

This document describes the protocol for an Internet Relay Chat (IRC) service. A virtual text chat room is managed by a central server, where individual clients can send messages to the server to be propagated to all other users in the same chat room, thus creating the illusion that they are all in the same room directly communicating with each other.

The rooms are controlled by the clients by sending requests to the server. A client can create a room, join a room, leave a room, and list all the available rooms. Rooms with no clients are automatically removed.

2. Conventions Used

The phrases "room" and "chat room" are used interchangeably and refer to the virtual text chat rooms that clients can join and chat in.

This document uses some keywords (in all capital letters) to mean the following:

MUST / REQUIRED: the functionality is absolutely required

MUST NOT: the functionality is absolutely prohibited

SHOULD / RECOMMENDED: there is a good reason to include the functionality, but it is not absolutely required

SHOULD NOT / NOT RECOMMENDED: there is a good reason to exclude the functionality, bit is not absolutely prohibited

MAY / OPTIONAL: the presence of the functionality is optional and up to the vendor

These definitions are paraphrased from RFC 2119 (see RFC 2119 for more precise definitions).

3. Basic Information

The connection will be through Port 5002, and the primary protocol will be TCP/IP. To provide a general overview of the initiation process, the client connects to the server and maintains a persistent session, which either the client or the server can terminate at any time.

Furthermore, it is important to note that the nature of the client-server communication is asynchronous.

4. Message Infrastructure

4.1 Message Format

All communications between the server and client(s) are sent as text strings with a maximum size of 1024 bytes. Data fields are embedded within the string, delimited by a separator token.
The basic message structure is:
`to_send = {opcode}+{separator token}+{message contents}`
For example, a client request to create a room is formatted as follows:
`to_send = {OPCODE_CREATE_ROOM}+{separator token}+{room_name}`
Which resolves to, `"2<SEP>minecraft_enthusiasts"`

4.2 Opcodes

As shown in 4.1, each packet will contain an opcode. The opcodes match to the following operations:

Code	**Operation**
0	Keep alive
1	Set username
2	Create room
3	List rooms
4	List my rooms
5	Join room
6	List members in room
7	Leave room
8	Send message
-1	Negative opcodes are reserved for errors

4.3 Error Messages

An error message with a negative opcode SHOULD be sent to the client when the server detects an error.

Code	Error type	Action
-1	Any error detected by server	Send the client a message with the error code and description of error.

4.4 Keepalive messages

Keepalive messages allow the server and client to know that they are still connected.

Both the client and the server MUST send a keepalive message every 5 seconds to notify the other end that they are still connected. If a client or a server hasn't received a keep alive message for 30 seconds or more, the connection SHOULD be terminated.

5. User and Room Names

User and room names follow the same rules. Each client sets their own username when they join the server and room names are set by the client when they create a room.

User and room names must meet the following requirements:

- Name must contain only alphanumeric characters and underscores
- Name must be unique

If any of these rules are broken, an error message will be returned to the client.

6. Client Messages

6.1 First message sent to the server

The client IS REQUIRED to initiate a connection with the server by making clear the protocol version intended for use, as well as provide a chat name. The server IS REQUIRED to react by terminating the connection to the client or match the socket of the user with the chat name.

6.2 Listing rooms

The client sends this to request the names of all of the active chat rooms. The server MUST respond back with the list of names.

6.3 Joining and creating rooms

6.3.1 Usage

The client sends a request to join a specific chat room; the message is sent to the server, and if there is a chat room with the specified name available (has at least one user), then the server is REQUIRED TO connect the client to that room.

If the specified chat room name is not found to be an active chat room, then the server IS REQUIRED TO inform the client of the error.

Conversely, a request to create a room requires that the room name is not already in use.

6.3.2 Field Definitions

room_name: This is the name of the room that the requesting client wants to join.

6.4 Leaving a room

The client will send a request to the server to be removed from a chat room; upon request, the server IS REQUIRED TO remove the client.

7. Server Messages

7.1 Listing rooms response

This is a simple message that is sent by the server to the clients. The message consists of information about the current list of rooms, and users.

7.2 Forwarding message to all clients in the room

7.2.1 Usage

The server MUST forward the message to all clients in the room.

7.2.2 Field Definitions

Field	**Definition**
Opcode	OPCODE_SEND_MESSAGE
sender	The name of the sender.
msg	The message being sent; MUST conform to basic guidelines for messages.
room_name	The name of the room to which the message was sent.

8. Error Handling

Client and Server MUST detect a broken connection and react as follows:
The Server MUST remove the client from all chat rooms; the Client MAY attempt to automatically reconnect.

9. Extra Features Supported

Clients are given a randomized color to distinguish their messages.

10. IANA Considerations

None.

11. Security Considerations

Messages are not encrypted, and are visible to the server.
Messages are vulnerable to any 3rd party capable of intercepting network traffic.

Authors' Addresses

Marten Sova	marten@pdx.edu
Timothy Shichalin	timofey2@pdx.edu
Rajat Kulkarni	rk24@pdx.edu