CS494 Hossam Elsamanoudy
Internet Draft Portland State University
Intended status: IRC Class Project Specification June 8, 2018
Expires: December 2018

Internet Relay Chat Class Project
draft-irc-pdx-cs494-00.txt

Status of this Memo

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

This specification describes a simple Internet Relay Chat (IRC) protocol by which clients can communicate with each other. This system employs a central server which ''relays'' messages that are sent to it to other connected users.

Users can join rooms, which are groups of users that are subscribed to the same message stream. Any message sent to that room is forwarded to all users currently joined to that room.

Users can also send private messages directly to other users.

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

All communication described in this protocol takes place over TCP/IP, with the server listening for connections on any port of the user’s choice. Clients connect to the same 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 inherently asynchronous - the client is free to send

messages to the server at any time, and the server may asynchronously send messages back to the client.

As is described in [4.2], both the server and client may terminate the connection at any time for any reason. This is specified by a Quit String sent to each of them.

Errors are handled to make up for any problem resulting from a server crash or a client crash

4. Message Infrastructure

4.1 Generic Message Format

All messages are sent through the socket.sendall() method. This is done through the TCP connection initiated between the server and its clients

4.1.1 Field Definitions

Prefix followed by body message
Prefix is checked to make sure that the message is not the first message between the server and the client. If this is the case, the client is added into the server's database. Otherwise, the messages are handled

4.2 Terminating Message

A quit string is specified as follows:

```
QUIT_STRING = '<$quit$>'
```

This is checked at both sides to know when one of them wants to close the connection.

4.2.1 Usage

MAY be sent by either the client or the server prior to closing the TCP connection to inform the other party that the connection is closed intentionally not for a connection error that terminated the connection without an intention. If either client or server receives this message, that entity SHOULD consider the connection as terminated.

4.2.2 Field Definitions

Fields are the same as the general message format. Here the message body contains the quit string.

4.3 Keepalive Messages

In python, the initiated TCP connection between a server and a client already takes care of this issue.

Setblocking was set to zero. Explanation is as follows. If a recv() call doesn’t find any data, or if a send() call can’t immediately dispose of the data, an error exception is raised; in blocking mode, the calls block until they can proceed. s.setblocking(0) is equivalent to s.settimeout(0.0).

4.3.1 Usage

Provides an application-layer notification of a disconnected peer.

Both client and server SHOULD watch for these keepalive messages and consider the other party disconnected if more than some set period of time has elapsed.

5. Label Semantics

Identifying both users and rooms involves sending and receiving labels. All label rules are the same, and MUST be validated as follows:

- Field size for transmission is always not more than 4096 as specified in recv_size
- Must consist entirely of readable ASCII character values, between 0x20 and 0x7E.
- If any of these rules are broken, the receiver MUST terminate the connection and MAY provide the terminating message.

6. Client Messages

6.1 First message sent to the server

The message contains the prefix “name” followed by the message body/user name entered by the client. This is ensured by the client program to ensure such identification will be understood by the server.

6.1.1 Usage

Before subsequent messages can be sent, a connecting client MUST provide a chat user name.

The server MUST associate the client's chat_name with the socket connection of the user. This message SHOULD be sent only once; if the server receives the message more than once, the server MAY ether ignore the message or terminate the client's connection.

6.1.2 Field Definitions

The message may contain any name. However, a word “name” prefix is added to identify the message when received by the server.

chat_name specifies the name the connecting client wishes to use. MUST follow label semantics.
chat_name must not be the same name provided by any other currently connected client. If the name already exists, the server will terminate the connection.

6.2 Listing Rooms

The message is started with the prefix <listrooms> with no body. The server replies with the list of rooms and the number of players each.

6.2.1 Usage

Sent by the client to request a list of all of the rooms currently occupied by at least one other client.

6.2.2 Response

If there are no rooms, the client SHOULD make a room first. Otherwise, a list of all the rooms along with the number of players is sent back to the requesting client.

6.3 Joining and Creating Rooms:
The message is started with the prefix <join> followed by a bunch of specified rooms to which the client wishes to connect.

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 inform all the users in the same chat room that this new user joined the room.

Every time the room's user list changes the server MUST send a message to all users in the room informing them of the new room membership.

6.3.2 Field Definitions

- room_name - Name of the room to join or create. MUST follow label semantics.

6.4. Leaving a Room
The message is started with the prefix <quit> followed by the room name to which the client wishes to disconnect.

6.4.1 Usage

Sent by the client to leave a chat room.

Upon receiving this message the server MUST remove the client from the specified room and MUST send a message to all users currently in that room to alert them that the user list has changed.

The server SHOULD ignore leave requests when the client is not currently a member of the specified room. In this case the client is sent back a message to indicate that he is not in such room.

6.4.2 Field Definitions

- room_name - Name of the room to leave. MUST follow label semantics.

6.5 Sending Broadcast Messages

The message is started with the prefix <send> followed by the number of selected rooms to send the message to followed by the room names to which the client wishes to send the message to. After that comes the message body which is sent to the chat rooms.

6.5.1 Usage

Sent by a client to send a text message to the entire a room.

The number of specified rooms are checked and if they exceed the number of rooms in which the client exists or if a room_name is invalid, the client is notified by this mistake and is asked to have another attempt.

6.5.2 Field Definitions

- target_number- Number of chat rooms to send the message to
- target_name - Name of the rooms to send the message to.
- msg - Message to send to the room.

Messages MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).

MUST be NULL terminated.

MUST not contain extra NULL terminators within the payload, they may only be at the very end.

If the message breaks any of these rules, the server MUST terminate the connection.

6.6. Listing Players

The message is started with the prefix <listplayers> followed by the room name to which the client wishes to know the players in. No body for the message.

6.6.1 Usage

Sent by a client to list the players of a specific room.

The room name has to be valid> Otherwise, the client is notified about this issue. The user is allowed to ask about a room which he does not belong to.

If the room has no members, then an Empty Message is sent back to the client who made the request.

6.6.2 Field Definitions

- target_name - Name of the room to list its players/members.

6.7 Sending Private Messages

The message is started with the prefix <priv> followed by the usename to which the client wishes to send the message to. After that comes the message body which is sent to the chat rooms.

6.7.1 Usage

Sent by a client to send a text message to a specific user in a shared chat room.

The name of the user is checked. If it is invalid, the requesting client is notified. Or if both users are not of a shared room, the requesting client is also notified of this issue by a 'You should only use a valid name who is in one of ur rooms' message.

6.7.2 Field Definitions

- target_name - Name of the user to send the message to.
- msg - Message to send to the room.

Messages MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).

MUST be NULL terminated.

MUST not contain extra NULL terminators within the payload, they may only be at the very end.

If the message breaks any of these rules, the server MUST terminate the connection.

6.8 Inquiry about Instructions

The message is started with the prefix <manual> with no body. The server replies with the list of client messages that are allowed to do different operations from the client side.

6.8.1 Usage

Sent by the client to request a list of all of the commands that he can send to the chat server and get a meaningful response.

6.8.2 Response

The server simply lists all the client messages that could be sent. In particular, the server informs the client of all the message prefixes that he can use to become able to perform any desired operation.

7. Server Messages

7.1. Listing Responses

This type of messages is directed to the requesting socket. This could be either a list of players for a specified room. Or it could also be a list of rooms with the players of each.

7.1.1 usage

Generic listing response message sent by the server to inform the client of a list. Used for both listing rooms with their listing users in a room.

7.1.2 Field Definitions

- item_names - Array of item names (players or rooms with the number of players), MUST follow label semantics.

7.2. Forwarding Messages to Clients

This type of messages are sent with a prefix of the sender name followed by a column followed by the message body sent.

7.2.1 Usage

One usage is that this is a forwarded message from the server indicating that the specified msg was posted to a room or many rooms the client is joined to. Server MUST set the target_name field to the name of the room(s) to which the message belongs.

Another possible usage is that this is a forwarded private message from another user intended specifically for the recipient. Server MUST set the target_name field to the name of the intended recipient.

7.2.2 Field Definitions

- sending_user - Name of the user who sent the message
- msg - Message sent to the room.

MUST consist entirely of readable ASCII character values, between 0x20 and 0x7E, and additionally can contain the carriage return and newline characters (0x0D and 0x0A).

8. Error Handling

Both server and client MUST detect when the socket connection linking them is terminated, either when actively sending traffic or by keeping track of the heartbeat messages (this is in fact assured by Python socket library). 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 client detects that the connection to the server has been lost, it MUST consider itself disconnected and will prompt the user that the Serve is down and he has the choice to reconnect.

9. "Extra" Features Supported

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

10. Conclusion & Future Work

This specification provides a generic message passing framework for multiple clients to communicate with each other via a central forwarding server.

Without any modifications to this specification, it is possible for clients to devise their own protocols that rely on the text-passing system described here. For example, transfer of arbitrary binary data can be achieved through trans-coding to base64. Such infrastructure could be used to transfer arbitrarily large files, or to establish secure connections using cryptographic transport protocols such as Transport Layer Security (TLS).

11. Security Considerations

Messages sent using this system have no protection against inspection, tampering or outright forgery. The server sees all messages that are sent through the use of this service. 'Private' messaging may be easily intercepted by a 3rd party that is able to capture network traffic. Users wishing to use this system for secure communication should use/implement their own user-to-user encryption protocol using SSL/TLS or

depending on RSA encryption techniques that could also be added to Python script.

12. IANA Considerations

None

12.1 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.

13. Acknowledgments

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

Authors' Addresses

Hossam Elsamanoudy
Portland State University Computer Science
1912 SW Ondine, Portland, OR 97201

Email: ehossam@pdx.edu