CS594 Armaan Roshani
Internet-Draft Portland State University
Intended status: IRC Class Project Specification June 8, 2018
Expires: December 8, 2018

# Internet Relay Chat Class Project
# RFC_final_rev.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 (broadcasted) to all users currently subscribed 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 port 7734. Clients connect to this 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. They MAY choose to send an error message to the other party informing them of the reason for connection termination.

The server MAY choose to allow only a finite number of users and rooms, depending on the implementation and resources of the host system.

## 4. Message Infrastructure

### 4.1. Generic Message Format

The format of Generic Messages is a string which can be encoded in up to 4k bytes.

### 4.2. Error Messages

Both the server and client may terminate their connection at any time for any reason. Both are “robust” in that they gracefully handle the other party terminating the connection.

The client outputs an Error Message: “Server down!” in the terminal when a loss of connection to the server is detected. This is because the server is generally assumed to stay up, so the user should be notified if it goes down. The server does not output an error message when a loss of connection to a client is detected, because it is assumed that clients will connect and disconnect often.

### 4.3. Keepalive Messages

The format of Keepalive Messages is determined by the Select system call. Both the server and the client rely on Python’s Select system call wrapper to establish and maintain TCP connections.

## 5. Identifiers and Constraints

o The server identifies rooms by keeping track of room names in a dictionary. It identifies users by keeping track of usernames and socket connections in lists and dictionaries. It keeps separate dictionaries of all users connected to the server, and all users connected to particular rooms. The server also keeps track of which room each client is currently broadcasting to by updating a current_room variable. The clients are agnostic of their own usernames or which room they are currently broadcasting to, though when their broadcast room is changed, the server sends that alert to be displayed to the user.

o The maximum field size for transmission is specified as 4KB.

o The messages must be at least 1 character. Leading whitespace is removed on the server side.

o If either of these rules are broken, the message is truncated or disregarded by the server. Depending on the instance, the server may pass an Error Message to its terminal.

## 6. Client Messages

The format of Client Messages is a 1B to 4KB datastream that is parsed as a string. The initial string is prepended with the identifier string “name:” which alerts the server that the word in the string is to be used to identify that specific client. All further strings from the client are parsed as follows:

o A prefix of “/rooms” requests the server to send the client a list of all available rooms. Any Additional Words in the client’s string will cause the server to return Usage Instructions to the client.

o A prefix of “/list” with no Additional Words requests the server to send the client a list of all users on the server.

o A prefix of “/list” with one Additional Word requests the server to send the client a list of all users in the room specified by the Additional Word. The server compares the Additional Word to its list of active rooms. If no such room is active, it returns an error that says as much to the client. More than one Additional Word in the client’s string will cause the server to return Usage Instructions.

o A prefix of “/join” or “/switch” with one Additional Word requests the server to add the client to the room specified in the Additional Word. The server compares the Additional Word to its list of active rooms. If no such room is active, it creates it. If

the client is not already added to it, it adds the client to it. It then sets that room as the client's broadcast room. More than one Additional Word in the client's string will cause the server to return Usage Instructions.

o A prefix of "/leave" with one Additional Word requests the server to remove the client from the room specified in the Additional Word. The server compares the Additional Word to its list of rooms the client is subscribed to. If the client is subscribed to no such room, it returns an error that says as much to the client. If the client is subscribed to the room specified, the server removes the client from that room's subscription list. If the client was set to broadcast to that room, it disabled the client broadcast until the client requests "/join" or "/switch" for another room. More than one Additional Word in the client's string will cause the server to return Usage Instructions.

o A prefix of "/msg" with at least two Additional Words requests the server to send a private message directly to another client. The server compares the first Additional Word to its list of clients. If no such client is on the server, it returns an error that says as much to the client. If the target client exists, the server removes the prefix and first Additional Word from the string, and forwards the rest of the string to the target client. Less than two Additional Words in the client's string will cause the server to return Usage Instructions.

o A prefix of "/help" will cause the server to return Usage Instructions.

o A prefix of "/quit" will cause the server to return a control word to the client that causes the client to exit gracefully.

## 7. Server Messages

The format of Server Messages is as follows:

o Usage Instructions:

```
/list - lists all users on server
/list [room name] - lists all users in a specific room
/join [room_name] - joins a room. If the room does not exist yet,
  it will be created.
/switch [room name] - switch which room you broadcast to
/leave [room_name] - leaves a room.
/msg [user] - Private Messages a user.
/help - shows instructions
/quit - ends the session
```

- Broadcast message from a room:

  [room_name] user_name: message

- Private message from another client:

  <user_name>: message

- Control word to cause client to exit gracefully:

  <$quit$>

## 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. 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 MAY choose to reconnect.

## 9. "Extra" Features Supported

Private messages between clients is implemented. The control sequence is detailed in the relevant part of 6. Client Messages.

## 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 transcoding to the (yet to be determined) format that messages will use. 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.

## 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 CS594SampleRFC.pdf as a template.

The IRC server, client, and backend structure was based loosely on a less feature-rich Python client/server system called pychat. Notable improvements include: multiple room subscriptions per client, client-to-client private messages, more robust error handling, more robust prefix and argument (called Additional Words above) handling, and stripped-down server and clients, in favor of consolidating functionality into the shared backend. That pychat system can be found here: https://github.com/xysun/pychat