minChat Class Project

Status of this Memo

This Internet-Draft is submitted 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 June 1, 2017.

Copyright Notice

Abstract

This memo describes the communication protocol for minChat, and IRC-like client/server chat messaging system. This is a project for the Internetworking Protocols (CS469) class at Portland State University.

1. Introduction

This specification describes a simple IRC (Internet Relay Chat) protocol called minChat. Clients can connect to a central server and communicate with each other via text messages. The relay in the IRC name refers to the central server that ‘relays’ messages between connected clients. Users can join rooms, and any client that messages in the room will have their message broadcasted to all other connected clients in that room.

2. Conventions used in this document

The keywords "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 keywords listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. Basic Information

The minChat server defaults to running on port 9999. This can be changed as necessary for different configurations and network needs. The client can use any port allocated by the operating system to connect to the server.

All data sent between the client and server is over the TCP/IP networking protocols. The client and server maintain a persistent TCP connection to exchange messages. The client sends messages over this pipeline, and the server can also push messages to the client via the same connection. The protocol is asynchronous, in that the client and server can communicate freely with one another whenever suitable.

The protocol is plain-text-based. Each command is one continuous line, ended with the CRLF (\r) character signaling the end of message. This is done for easy of programming both the client and server, and also for human readability and understanding. The minChat protocol aims to minify protocols such as the well-known IRC protocol. In this, it has been stripped to bare essentials for network efficiency, easy of use, and above all, simplicity.

The server or client may leave the connection at any time, with or without reason. The client or server MAY state why the connection is being terminated, but is NOT required to.

This protocol specification does not impose a “theoretical” limit or constraint on connected clients. Likewise, there is no limit imposed in this document for the amount of users in a given room. This is also dictated only by server resources, or system administrator configurations.

4. Message Structure

4.1. Generic Message Format

As mentioned in [3], the protocol message formats are a single plain-text line, ended with the CRLF (\r) ascii character. A generic client message format follows:

<command> <parameters>\r

A server message follows:

%<status code> <parameters> %<optional message>\r

4.1.1. Field Definitions

* status code - returned status code from the server. Status codes fall into 4 categories, defined in 4.1.2.
* parameters - a list of commands, or information to be passed along
* optional message - the server or client MAY send an optional message with the message, but it is NOT required.

4.2 Status Codes Introduction

The status codes for minChat are divided into four sections, inspired by HTTP status codes [RFC 2616].

1XX - informational
2XX - success
3XX - client error
4XX - server error

This allows for 99 status codes for each section, which is more than enough to implement the simple protocol in minChat.

4.2.1 Status Codes

1XX - informational

Reserved for future use.

2XX - success

```
OK_QUIT              = 201
OK_REG               = 202
OK_AUTH              = 203
OK_ROOMCREATE        = 204
OK_ROOMEXISTS        = 205
OK_ROOMLIST          = 206
OK_NEWJOIN           = 207
OK_LEAVE             = 208
OK_LIST              = 209
OK_JOIN              = 210
OK_MSG               = 211
OK_SEND              = 212
```

3XX - client error

```
ERR_BANNED           = 301
ERR_TIMEOUT          = 302
ERR_USERNAME_TAKEN   = 303
ERR_INVALID          = 304
ERR_NOUSER           = 305
ERR_DENIED           = 306
ERR_NOUSER           = 307
ERR_TOOBIG           = 308
ERR_NOCHAN           = 309
ERR_ALREADYINCHAN    = 309

4XX - server error

ERR_OOM              = 401
```

4.3. Keepalive Messages

```
Server sent:
%PING <unix epoch>\r

Client responds:
PONG <original epoch time sent in PING>\r
```

4.3.1 Usage

The client MAY respond to PING messages. If the client does not respond to PING messages within the server-specified amount of time, the server MAY send a DISC message and terminate the connection, or terminate the connection without the QUIT message. The keepalive time MUST NOT exceed 240 seconds.

5. Commands

5.1 QUIT

5.1.1 Usage

The QUIT command can be sent by the client to let the server know it is going to disconnect. The server then acknowledges the QUIT and then terminates the connection. This is not required, as the client can disconnect at any time, but rather a way to “nicely” end the connection.

5.1.2 Message Format

Client:
QUIT\r

Server:
%<status code> %<optional message>\r

5.1.3 Field Definitions

Status code - QUIT from server is OK_QUIT.
Optional message - Server MAY send a parting message if wanted.

5.2 DISC

5.2.1 Usage

The DISC command is sent by the server if the server wants to let a client know it is being disconnected. Reasons for this are ERR_BANNED (user has been banned from server) and ERR_OOM (server resources constrained, so kicking clients). Also, if the user does not respond to PING messages in time, it can also terminate the connection with ERR_TIMEOUT. It is up to the server programmer how to pick clients to disconnect if the server is resource constrained. The server is NOT required to send a DISC command before closing a connection, but it is nice to do so.

5.2.2 Message Format

Server:
%<status code> %<optional message>\r

5.2.3 Field Definitions

Status code - Can be either ERR_BANNED, ERR_TIMEOUT, ERR_OOM.
Optional message - Can be text telling client why they are being disconnected. This is not required.

5.3 REG

5.3.1 Usage

When the client wants to connect to the minChat server, the client is required to pick a username to use. The client MUST register a username with a password to use it.

5.3.2 Message Format

Client:
REG <username> <password>\r

Server:
%<status code> %<optional message>\r

5.3.3 Field Definitions

Username - desired username / nickname to use in minChat session
Password - password used to authenticate for username account
Status code - can be OK_REG, ERR_USERNAME_TAKEN, ERR_INVALID
Optional message - server may send message to client letting it know more about status code

5.4 AUTH

5.4.1 Usage

When a username is already registered, the client MUST send an AUTH message to gain access to the user account. Usernames must be registered with the REG command prior to using the AUTH command, or the client will receive ERR_NOUSER.

5.4.2 Message Format

Client:
AUTH <username> <password>\r

Server:
%<status code> %<optional message>\r

5.4.3 Field Definitions

Username - username of pre-existing account to authenticate with

Password - password stored in server for username account
Status code - ERR_DENIED (wrong password), ERR_NOUSER (user does not exist in registration), or OK_AUTH.
Optional message - server MAY send optional text message to elaborate on status code if warranted.

5.5 MSG

5.5.1 Usage

The MSG command is the heart of the minChat client. This command allows users to message channels.

Messages shall not exceed 512 characters in the <message> field. If this limit is exceeded, the server will send ERR_TOOBIG (message length exceeded). For messages longer than 512 characters, the message MUST be split into multiple minChat commands.

Sends message to all users in the channel message. Server echoes message to all clients in that room message, which includes the user who sent the message. The client is responsible for printing the sender's message in their client after confirmation received from the server.

5.5.2 Message Format

Client:
MSG #<channel> %<message>\r
MSG <username> %<message>\r
Server:
%<status code> <channel> <username> %<message>\r

5.5.3 Field Definitions

Status code - OK_SENT (message was successfully sent), ERR_INVALID (message format invalid), ERR_NOCHAN (no channel exists to send message to).
Username - user who sent message
Message - plain text message that is being sent by user

5.6 JOIN

5.6.1 Usage

A client can join a room to chat with other users. Channels are represented by #channel.

If the room is not created yet, the server responds with OK_ROOMCREATE to the client, and register the client in the room for future messages in that channel.

If the server already knows the room exists, it responds with OK_ROOMEXISTS, and then another message with OK_ROOMLIST with a comma separated list of users in that channel.

If the user enters an invalid room name (only alphanumeric and underscores allowed), the server will respond with ERR_INVALID.

If another user enters the same channel/room, the server will send a JOIN to all current room clients letting them know a new user joined the channel. This status code is OK_NEWJOIN.

5.6.2 Message Format

Client:

JOIN #<channel>\r

Server:

%<status code> <channel> %<optional message>\r
%<status code> %<optional message>\r

5.6.3 Field Definitions

Server:
Status code - OK_ROOMLIST, OK_ROOMEXISTS, ERR_INVALID, OK_NEWJOIN, OK_ROOMCREATE
Username/channel - when joined room successfully, username/channel is echoed back to client.
Optional message - server MAY send optional text message to elaborate on status code if warranted.

5.7 LEAVE

5.7.1 Usage

A user can leave a room at any point they would like. The server will then register that the user has left the room, and stop broadcasting messages for that channel to the user's client.

Status codes can be ERR_NOCHAN (no such channel to leave, or user is not part of the channel specified), ERR_INVALID (invalid input), OK_LEAVE (server echos that client has left the channel).

5.7.2 Message Format

Client:
LEAVE #channel\r
LEAVE %username\r

Server:
%<status code> %<optional message>\r

5.7.3 Field Definitions

Status code - ERR_NOCHAN, ERR_INVALID, OK_LEAVE.
Optional message - server MAY send optional text message to elaborate on status code if warranted.

5.8 LIST

5.8.1 Usage

The client may obtain a list of channels on the server by using the LIST command. The channels will be returned to the client in a comma separated format.

5.8.2 Message Format

Client:
LIST\r

Server:
%<status code> <list> %<optional message>\r

5.8.3 Field Definitions

Status code - ERR_NOCHAN (no channels to list), OK_LIST (list of channels follows)
List - comma separated list of channels on the server
Optional message - server MAY send optional text message to elaborate on status code if warranted.

5.9 MEMLIST

5.9.1 Usage

The client may obtain a list of users in a channel. This will only be allowed for channels the user is in.

5.9.2 Message Format

Client:
MEMLIST #channel\r

Server:
%<status code> <list> %<optional message>\r

5.9.3 Field Definitions

Status code - ERR_NOCHAN (not in channel), OK_LIST (list of users follows)
List - comma separated list of users ex: ben,bob,jill,hillary
Optional message - server MAY send optional text message to elaborate on status code if warranted.

6. Error Handling

Errors are recognized on the client side as either 3XX or 4XX status codes returned from the server. The client MUST assume that it will get server errors, and handle them accordingly in a graceful fashion.

The server must deal with malformed input without crashing or other undesirable issues. The server must NEVER trust the clients from a security perspective, and always be skeptical of their inputs. If a major error happens on the server, it should

try to notify clients with DISC commands before gracefully exiting. This is not a hard requirement, but would be better than killing all connections to the server without cause.

7. Security Considerations

minChat does not have any “baked-in” security implementation. All messages are in plain-text and are sent unencrypted over the Internet. If secure communications are required, the end user much encrypt data before transport.

8. Conclusion & Future Work

The features and commands of minChat could be added to at a later date as demand requires. The goal of the project is to have a barebones IRC style client and server model. Features will be considered, but only when absolutely necessary. This keeps with the rule of keeping minChat very simple, fast, and easy to program.

9. IANA Considerations

None

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2616] “Hypertext Transfer Protocol -- HTTP/1.1“, RFC2616, June 1999

Author’s Address

Ben Reichert
1900 SW 4th Ave
Portland, OR 97201

Email: breic2@pdx.edu