CS-594
Internet Draft
Intended Status: IRC Protocol
Expires: June 2022

Kelsey Werner
Portland State University
December 3, 2021

# Internet Relay Chat Protocol
Irc-protocol-rfc.pdf

## Status of this Memo

This Internet-Draft will expire on June 3, 2022.

## Copyright Notice

## Abstract

This document outlines a protocol for client-server communication to support a simplified version of Internet Relay Chat (IRC) with limited functionality.

## Table of Contents

## 1. Introduction

The specifications detailed in this document present a protocol to support a simplified version of Internet Relay Chat (IRC) with limited functionality. This protocol allows distributed clients to communicate with each through a central server. The central server receives messages from clients and forwards those messages to all intended recipient clients.

This protocol permits users to initiate, subscribe to, and unsubscribe from streams of messages broadcast from the central server. Users are also able to contribute messages to any stream they are subscribed to. The protocol lets any number of users subscribe to a stream, but the server implementation may set size constraints. A stream that is intended only for use between two clients is considered private messaging.

## 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.

## 3. Basic Information

In this protocol, clients connect to the server using TCP/IP over port 2787. The server is able to maintain multiple parallel threads to manage each client connection, so each client-server connection can persist until either the client or the server MAY choose to close the connection. This protocol also provides guidance to gracefully handle an unexpected close of a connection by either the client or server. Because the client-server communication channel remains open until the end of the session, messaging between the client and the server can be asynchronous.

## 4. Message Infrastructure

### 4.1. Generic Message Format

```
COMMAND:parameter(s):payload
```

#### 4.1.1. Field Definitions

- ❖ All fields are separated by a colon without any spaces.
- ❖ COMMAND
  Indicates the intent of the message. This field will dictate how the rest of the message is parsed and which actions should be taken by the client or server. All messages MUST have the COMMAND field populated.
- ❖ payload
  This is the body of the message. Some COMMANDs may not require a payload.
- ❖ parameter(s)
  The parameter field is required for some message types but MUST never be used in messages where it is not required. The specific content of the parameter field is dependent on the context of the message type.

#### 4.1.2. Command Semantics

- ❖ The COMMAND field MUST be from the list of Command Codes in Section 4.1.5. If a COMMAND is sent from outside of this list, then an UNRECOGNIZED_COMMAND error message SHOULD be returned.
- ❖ MUST NOT contain a colon or any spaces. Use of colons could result in a malformed message, and use of a space SHOULD result in a COMMAND_SPACE error message.

- ❖ MUST be formatted in snake_case.

- ❖ MUST be uppercase ASCII character values.

### 4.1.3. Parameter Semantics

- ❖ MUST NOT contain a colon or any spaces. Use of colons could result in a malformed message, and use of a space SHOULD result in a PARAMETER_SPACE error message.

- ❖ MUST be ASCII character values.

- ❖ SHOULD be formatted in snake_case.

- ❖ SHOULD be limited to 50 characters. Client or server MAY choose to return an OVERSIZED_PARAMETER error message.

### 4.1.4 Payload Semantics

- ❖ SHOULD be ASCII character values.

- ❖ SHOULD be limited to 500 characters. Client or server MAY choose to return an OVERSIZED_PAYLOAD error message.

### 4.1.5. Command Codes

- ❖ ERROR
- ❖ NAME
- ❖ STILL_ALIVE
- ❖ QUIT
- ❖ JOIN
- ❖ JOIN_RESPONSE
- ❖ LEAVE
- ❖ LEAVE_RESPONSE
- ❖ MESSAGE
- ❖ MESSAGE_USER
- ❖ USERS
- ❖ USERS_RESPONSE
- ❖ ROOMS
- ❖ ROOMS_RESPONSE

## 4.2. Error Message Format

```
ERROR:error_code:name:message
```

### 4.2.1. Field Definitions

- error_code
  A three digit integer from the error codes listed in Section 4.2.2.

- name
  Some error messages will include a user_name or a room_name, but it is not a required field for all error messages. MUST follow parameter semantics.

- message
  This is the body of the error messages. SHOULD follow payload semantics. Error Codes in Section 4.2.2 include a default message, but the implementation MAY replace this message with a custom message.

### 4.2.2. Error Codes & Conditions

- UNRECOGNIZED_COMMAND
  ERROR:100:Command is not included in the list of approved commands

- OVERSIZED_PARAMETER
  ERROR:101:Parameter has exceeded allowed value of 50 characters

- OVERSIZED_PAYLOAD
  ERROR:102:Payload has exceeded allowed value of 500 characters

- COMMAND_SPACE
  ERROR:103:Command contains spaces

- PARAMETER_SPACE
  ERROR:104:Parameter contains spaces

- DUPE_USERNAME
  ERROR:105:Username already in use

- UNREGISTERED_CLIENT
  ERROR:106:Client not registered with server

- UNKNOWN_ROOM
  ERROR:107:room_name:This chat room does not exist

- UNAUTHORIZED_ROOM
  ERROR:108:room_name:User is not a member of this chat room

- UNKNOWN_USER
  ERROR:109:user_name:This user does not exist

## 5. Messages

### 5.1. One-Way Handshake

```
NAME:user_name
```

#### 5.1.1. Usage

After the TCP/IP connection has been established, this message MUST begin application level communication between a client and the central server. This message is sent from the client to register the client with the server using a unique username. If an unregistered client tries to send any other message type to the central server, then they MUST receive an UNREGISTERED_CLIENT error message. The client SHOULD NOT wait for a response from the server because the STILL_ALIVE message exchange will provide feedback to the client that the server is (or is not) connected (see Section 5.2).

#### 5.1.2. Field Definitions

- user_name
  This is the unique username used to identify the socket for a specific client-server TCP/IP communication channel. If there is an active client connection using a distinct user_name, then other clients MUST NOT use this identifier as long as the connection is active. If another client sends a NAME message with a duplicate user_name, they MUST receive a DUPE_USERNAME error message from the server.  MUST use parameter semantics.

## 5.2. Keep Alive

```
STILL_ALIVE
```

### 5.2.1. Usage

The exchange of STILL_ALIVE messages between the clients and central server allows the persistence of the communication channel and ensures a graceful failure if a host crashes. Every five seconds, each client MUST send a STILL_ALIVE message to the server, and the server MUST send a STILL_ALIVE message to every client every five seconds. Each host SHOULD listen for STILL_ALIVE messages, and if a still alive message is not received within an implementation-specified time window, then the connection SHOULD be considered terminated. The timeout SHOULD NOT exceed 15 seconds.

## 5.3. Close Connection

```
QUIT
```

### 5.3.1. Usage

This message MAY be sent from the server to notify a client that it is closing the TCP/IP connection. It MAY also be sent by a client to the server to request that the server close the TCP/IP connection. It is not strictly required to send this message as the STILL_ALIVE message exchange will alert hosts to a closed connection.

## 5.4. Create and Join Chat Room

### 5.4.1. Client Message

```
JOIN:room_name
```

#### 5.4.1.1. Usage

This message is sent from the client to the server when the client either wants to either create a new chat room or join an existing chat room. The client is not required to wait for a response from the server confirming success.

#### 5.4.1.2. Field Definitions

- room_name
  The name of the chat room that the client wants to create or join. If the client submits a room_name that does not already exist, then a new room will be created, and the client’s username MUST be added to the room’s list of users. If the client submits a room_name that already exists, then the user MUST be added to the room’s list of users. If the client submits a room that they are already subscribed to then the server SHOULD NOT send an error message. MUST use parameter semantics.

### 5.4.2. Server Message

```
JOIN_RESPONSE:room_name
```

#### 5.4.2.1. Usage

This message is sent from the server to the client in response to the JOIN message to confirm that the JOIN message was received. Depending on the specific JOIN request made by the client, JOIN_RESPONSE also indicates that a new room was created or that the username was added to the room list of users. This message is only sent to one client and is not broadcast to all clients connected to the server because the server is the source of truth for the most updated list of rooms and users per room. Clients MUST use the USERS and ROOMS commands to pull this data from the server.

#### 5.4.2.2. Field Definitions

- room_name
  The name of the chat room that the client requested to create or join. MUST use parameter semantics.

## 5.5. Exit Room

### 5.5.1. Client Message

```
LEAVE:room_name
```

#### 5.5.1.1. Usage

Message sent from the client to the server to indicate that the user wants to unsubscribe from a room. The TCP/IP connection will still persist, but this client will no longer receive messages that are broadcast to this room from the server. The client is not required to wait for a response from the server confirming success.

#### 5.5.1.2. Field Definitions

- room_name
  The name of the chat room that the user wants to leave. If an unknown room_name is sent or the user isn’t already a member of the chat room, then the server SHOULD NOT send an error message. MUST use parameter semantics.

### 5.5.2. Server Message

```
LEAVE_RESPONSE:room_name
```

#### 5.5.2.1. Usage

This message is sent from the server to the client in response to the LEAVE message to confirm that the LEAVE message was received. The server MUST remove the username associated with this connection from the requested room’s list of users when the LEAVE request is received, and the LEAVE_RESPONSE message also confirms to the client that the user has been successfully unsubscribed from messages from this room.

#### 5.5.2.2. Field Definitions

- room_name
  The name of the chat room that the client requested to unsubscribe from. MUST use parameter semantics.

## 5.6. List Rooms

### 5.6.1. Client Message

```
ROOMS
```

#### 5.6.1.1. Usage

This message is sent from the client to request a list of all chat rooms from the server. This data is pulled from the server by client request rather than broadcast by the server whenever there is a change to the list of rooms.

### 5.6.2. Server Message

```
ROOMS_RESPONSE:rooms_list
```

#### 5.6.2.1. Usage

This message is sent by the server in response to a client’s ROOMS request message. It returns a space-separated list of all rooms stored on the server.

#### 5.6.2.2. Field Definitions

❖ rooms_list
MUST use payload semantics with the additional requirement that this is a space-separated list of all room names stored on the server. SHOULD NOT respect the length requirement of payload semantics. Spaces MUST only be used to distinguish between room names, and each individual room name in the list MUST use parameter semantics. If no rooms exist, then the rooms_list field MUST be a single space.

## 5.7. List Users

### 5.7.1. Client Message

```
USERS:room_name
```

#### 5.7.1.1. Usage

This message is sent from the client to request a list of all users in a room from the server. This data is pulled from the server by client request rather than broadcast by the server whenever there is a change to the list of users in the room.

#### 5.7.1.2. Field Definitions

- room_name
  The client is requesting to receive the list of users in the room that has the room name supplied in this field. If there is no room with the name room_name, then the client MUST receive an UNKNOWN_ROOM error message. MUST use parameter semantics.

### 5.7.2. Server Message

```
USERS_RESPONSE:room_name:users_list
```

#### 5.7.2.1. Usage

This message is sent by the server in response to a client’s USERS request message. It returns a space-separated list of all users stored on the server for the specific room requested by the client.

#### 5.7.2.2. Field Definitions

- room_name
  The name of the chat room that the client requested to retrieve the list of users from. MUST use parameter semantics.

- users_list
  MUST use payload semantics with the additional requirement that this is a space-separated list of all user names stored on the server for the room_name room. SHOULD NOT respect the length requirement of payload semantics. Spaces MUST only be used to distinguish between user names, and each individual user name in the list MUST use parameter semantics. If chat room does not have any members, then the users_list field MUST be a single space.

## 5.8. Chat Room Messages

### 5.8.1. Client Message

```
MESSAGE:room_name:message_body
```

#### 5.8.1.1. Usage

This message is sent from the client to the server when the user wants to post a message to a chat room.

#### 5.8.1.2. Field Definitions

- ❖ room_name
  This field indicates the name of the chat room that the user wants to post a message to. If the client submits a room that does not exist, then the server MUST return an UNKNOWN_ROOM error message. If the client is not a member of this room, then the server will return an UNAUTHORIZED_ROOM error message, and the message_body MUST not be posted to the chat room. MUST use parameter semantics.

- ❖ message_body
  This is the text message that the user wants to post to the chat room. MUST use payload semantics.

### 5.8.2. Server Message

```
MESSAGE:room_name:user_name:message_body
```

#### 5.8.2.1. Usage

This message is broadcast in response to the client-side MESSAGE request. The server forwards the message_body sent by the client to all other clients that are subscribed to the room_name room. This message SHOULD also be returned to the original sending client as a confirmation that their text chat message was successfully broadcast. The sending client SHOULD wait to receive this confirmation before posting the message_body to the room locally to help ensure the user does not mistakenly believe a message was successfully broadcast in case of a failure.

#### 5.8.2.2. Field Definitions

- ❖ room_name
  This field indicates the name of the chat room. The receiving client can use this field to determine which room to post the message_body to. MUST use parameter semantics.

- user_name
  This field indicates the name of the user that sent the message. The receiving client can use this field to determine which username to attribute the message_body to when it’s posted to the chat room. MUST use parameter semantics.

- message_body
  This is the text message for the receiving clients to post to the chat room. MUST use payload semantics.

### 5.9. Private User-to-User Message

#### 5.9.1. Client Message

```
MESSAGE_USER:user_name:message_body
```

##### 5.9.1.1. Usage

This message is sent from the client to the server when the user wants to post a private message directly to another user (rather than to a public group chat room).

##### 5.9.1.2. Field Definitions

- user_name
  This field indicates the username of the user that the sending user wants to send a direct message to. If the client submits a user_name that is not for an active user, then the server will return an UNKNOWN_USER error message. MUST use parameter semantics.

- message_body
  This is the text message that the user wants to send to another user. MUST use payload semantics.

#### 5.9.2. Server Message

```
MESSAGE_USER:target_user_name:sending_user_name:message_body
```

##### 5.9.2.1. Usage

This message is sent in response to the client-side MESSAGE_USER request. The server forwards the message_body sent by the client to the client identified by the target_user_name parameter. This message SHOULD also be returned to the original sending client as a confirmation that their text chat message was successfully forwarded. The sending client SHOULD wait to receive this confirmation before posting the message_body in its local feed to help ensure the user does not mistakenly believe a message was successfully forwarded in case of a failure.

#### 5.9.2.2. Field Definitions

- target_user_name
  This field indicates the name of the user that is receiving the message_body. This can be used by both the sending and receiving client to display the message exchange to the text feed. MUST use parameter semantics.

- sending_user_name
  This field indicates the name of the user that sent the message_body. The receiving client can use this field to determine which username to attribute the message_body to when it’s posted to the text feed. MUST use parameter semantics.

- message_body
  This is the text message for the receiving client to post to the text feed. MUST use payload semantics.

## 6. Host Crash Management

### 6.1. Client Crash

If a client crashes, the server program will continue to run, accept new client connection requests, and maintain existing client connections. The server can detect that a client has crashed if they don’t receive a STILL_ALIVE message from that client within the implementation-specific time window. Additionally, the server program MUST use the exception handling capabilities provided by the programming language chosen for implementation. The entire body of the server program SHOULD be included within the scope of the exception handling, and if the program is multithreaded, each thread SHOULD be wrapped in at least one exception handler of its own. All threads dedicated to a specific client MUST stop execution if one thread detects that this client has crashed, but the primary server thread listening for client connection requests MUST not stop execution if one client crashes. If a client crash is detected, the server SHOULD output the error details to the terminal, so that there is a log of any crashes. Finally, a crashed client MUST be removed from the list of client TCP connections, and that client’s user name MUST be removed from the members list of any chat rooms where they are a member.

### 6.2. Server Crash

If the server crashes, all clients connected to the server MUST terminate their connections and shut down their programs. Before ending the program, the client SHOULD output a message to the user to let them know that an error has occurred. Clients can detect that the server has crashed if they don’t receive a STILL_ALIVE message from the server within the implementation-specific time window. Additionally, the client program MUST use the exception handling capabilities provided by the programming language chosen for implementation. The entire body of the client program SHOULD be included within the scope of the exception handling, and if the program is multithreaded, each thread SHOULD be wrapped in at least one exception handler of its own. All threads MUST stop execution if one thread detects that the server has crashed.

## 7. Extra Supported Features

Beyond the minimum specifications outlined in the project requirements, two additional features are supported by this protocol.

### 7.1. Private Messaging

Users can directly message each other without joining a public room.

### 7.2. Cloud Connected Server

The server implementation of this protocol is cloud hosted on Amazon Web Services.

## 8. Conclusion & Future Work

As is, this protocol provides specifications for the sending of asynchronous text messages between clients through a central server. This protocol could be extended to allow for file transfer through messages. Additionally, this protocol does not provide any security guarantees, so it could also benefit from future improvements for secure messaging.

## 9. Security Considerations

This protocol does not provide any secure messaging or encryption. All messages are visible while they are in transit over TCP/IP and while they are being stored on the central server. Security considerations are left as a future enhancement and are currently the responsibility of the implementing entity.

## 10. IANA Considerations

None

### 10.1. 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.

## 11. Acknowledgements

This document was prepared using Google Docs without a template.