COMP 4985 Protocol Team
Category: Standards Track

Blaise Klein
Kevin Nguyen
Oceaan Pendharkar
BCIT
April 2025

Chat Protocol

Abstract

This document specifies the chat room protocol developed by the COMP 4985 Protocol Team. It defines the structure and behaviour of client-server communication, including user authentication, message transmission, and server management. The protocol is designed to facilitate real-time communication while maintaining a lightweight, efficient binary format for message exchange. This document outlines the fundamental components of the chat system, including the client, server, and server manager, as well as the interactions between these components. Additionally, it provides guidance on message formats, connection procedures, and security considerations. Feedback and discussion on improvements to this protocol are encouraged.

Status of This Memo

This document specifies a chat room protocol for the Internet community and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Copyright Notice

Table of Contents

1 . Introduction

The objective of the Chat Message Protocol is to manage chat servers, and have those chat servers manage client connections and chat messages. This specification describes the protocol and required systems for clients to connect to a chat server, and send and receive messages between each other. The functionality is split between three systems, the server manager, a server starter and chat server pair, and some number of clients. Clients are required to create and authenticate into an account to send and receive chat messages, which are broadcasted by a chat server.

2. Requirements Language

2.1 Document Convention

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 [BCP14]. As each of these terms was intentionally and carefully chosen to improve the interoperability of the chat protocol, each use of these terms is to be treated as a conformance requirement.

3. Security Consideration

3.1. Reliance on TCP

The protocols must be implemented using TCP connections instead of UDP connections for various reasons. Primarily TCP is used to ensure that messages are received properly. TCP is also used to ensure maintained connection from client to server, and from server to server manager. This will be noted later as well when talking about authentication and user connection.

3.2. Authentication

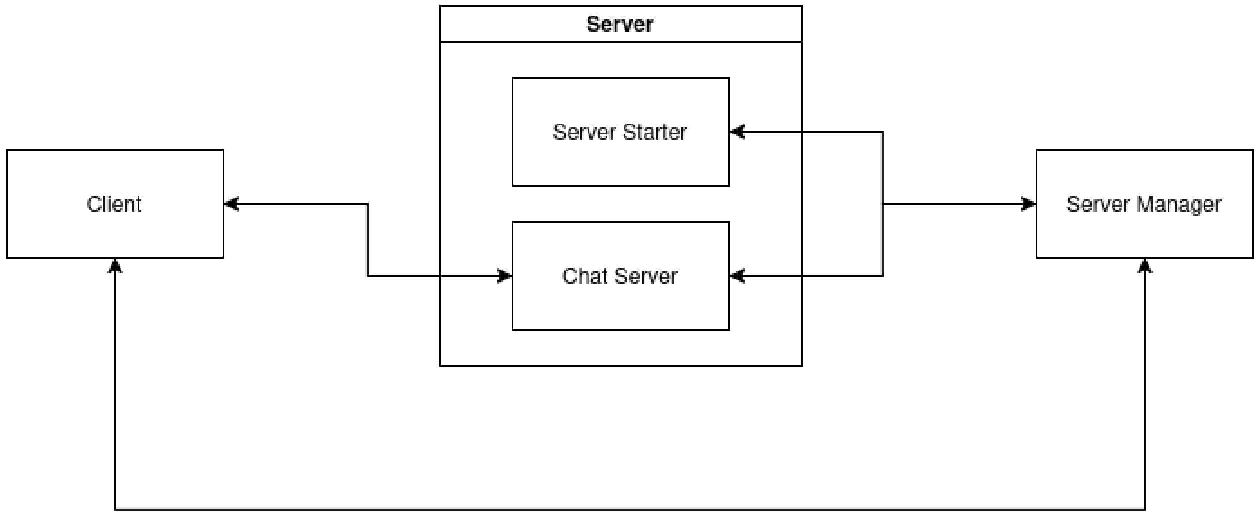
Users must authenticate a client when attempting to access a chat server. To ensure that clients are not able to log into another user's account, usernames must be unique to aid in ensuring that user accounts cannot be spoofed. Confirmation on what part of the login request was wrong, either the username or password, should not be done to reduce the amount of knowledge given to users attempting to brute force a password. Additionally, as user information must be stored, it could be implemented to obfuscate the passwords of users to the server maintainers.

3.4. Server Spoofing

It is possible that a server may request to connect to the server manager with malicious intent to gather user data. To ensure that this does not happen, a simple solution is that the servers and server manager be maintained by one body and that connection be made on a secure IP address and port that only the server and server managers know. Ultimately it should be ensured that clients are not given the IP address and port of a spoofing chat server which could have malicious intent.

4. Overview of Systems

4.1 Basic Structure



The chat protocol requires 3 main systems, a server, a server manager, and some number of clients. This document details three protocols required for the following communications.

- Client and server communication
- Client and server manager communication
- Server and server manager communication.

The server has two subsystems, a server starter and a chat server. The server starter initially exists and must connect to a server manager. The server manager should only ever have one server starter connected at a time. Once connected, the server starter can receive requests to start or stop a single chat server instance. Once started the chat server can communicate with the server manager to send monitoring information, specifically the number of online users, and the number of messages sent since the chat server started. The chat server can also accept multiple clients, manage user creation, manage user login and logout, receive messages, and broadcast received messages to online users. The port used for client connections must be hard coded, and must be the same value set by the server manager that it is connected to.

The server manager manages both the server lifecycle, and the initial connection of clients. It handles server starter connections and can start or stop chat servers via this connection. Only one chat server can be online at once, and the server manager must be able to handle receiving monitoring information about the chat server, including the number of online clients, and the number of chat messages sent this session. The server manager must also be able to handle the initial connection of clients. This entails receiving client connections, and forwarding the current active chat server’s IP address and port number to receive data on. The IP address will be the same IP address that the server starter is connected to, and the port will be a hard coded value shared by the server starter and server manager.

The client must be able to connect to an active server, send chat messages to the chat server, and receive messages from the chat server. Connecting to the active server requires two stages, connecting to a server manager and requesting the current chat server's IP address and port number, and connecting to the chat server itself. Once connected the client must request to either create a user, or login to an existing user account with a username and password. Once logged in, it will be able to send and receive messages from the chat server. The client must be able to logout from the server and gracefully close its connection.

5. Binary Format Rules

5.1. ASN.1 Structure

The packet and data structures of this protocol are documented following ASN.1 definition standards as defined in X.680 [X680].

Example:

```
Name ::= SEQUENCE {
    field    Type [options] (size)
    field    Type [options] (size)
}
```

5.2. Custom Encoding Rules

The chat protocol encoding rules are based on Basic Encoding Rules (BER) as defined in X.690 [X690]. Several changes were made to BER to accommodate the development of the chat programs. Deviations from BER are listed in Section 5.2.4.

Packets are split into two sections, header and payload, and differ in encoding.

5.2.1. Header Encoding

Headers are encoded using only the value of each field. Set sizes for each field are specified in each protocol. The values are packed together with no delimiter.

5.2.2. Payload Field Encoding

Each field within a payload must be encoded following this format:

> [Type] [Length] [Value]
>
> [Type] The type of the field, encoded in the corresponding Tag in Section 5.2.3.
> [Length] The length of the value of the field in bytes. This is limited to one byte.
> [Value] The value of the field encoded into bytes.

5.2.3. Tag Encoding

Field Types MUST be encoded using one byte with the following encoding tags:

Tag (decimal)	Tag (hex)	Type
2	02	INTEGER
5	05	NULL
10	0A	ENUMERATED
12	0C	UTF8String
24	18	GeneralizedTime

GeneralizedTime is encoded using Zulu time: YYYYMMDDhhmmssZ

5.2.4. Deviations from BER

- The Sequence tag is not encoded and only used for documentation purposes
- The encoded Length of each field is restricted to one byte
- The encoded Type of each field is restricted to one byte and only those found in Section 5.2.3
- GeneralizedTime includes the full four-digit value for year instead of two

5.2.5. Field Encoding Examples

The following encoded examples are displayed with hexadecimal values.

`INTEGER: x = 4985`

Type	Length	Value
02	02	13 79

`NULL:`

Type	Length	Value
05	00	-

`ENUMERATED: x = SYS_SUCCESS (0)`

Type	Length	Value
0A	01	00

`UTF8String: x = ‘hello’`

Type	Length	Value
0C	05	68 65 6C 6C 6F

`GeneralizedTime: x = 2025 March 20 12:30:00 UTC`

Type	Length	Value
18	0F	32 30 32 35 30 33 32 30 31 32 33 30 30 30 5A

5.3. Version Number

The three protocols detailed later in this document all require a version number value that must be set according to the version of the protocol

As of this version of the RFC the version value must be set to the number 3.

6. Chat Message Protocol

6.1. Overview

The Chat Message protocol is used to communicate between client and server programs. Each packet is split into a header and payload.

6.2. Header

The header holds information regarding the program, the sender, and the payload.

6.2.1. Header Structure

The header has a set size of six bytes with four fields:

```
Header ::= Sequence {
     packet_type ENUMERATED (1 BYTE)
     version INTEGER (1 BYTE)
     sender_id INTEGER (2 BYTES)
     payload_len INTEGER (2 BYTES)
}
```

[packet_type] indicates to the receiver what kind of information is being sent
[version] the protocol version that the sending program is using (currently supported: 3)
[sender_id] id of users set by the server. Server-specific packets use an id of 0.
[payload_len] total length of the payload in bytes.

6.2.2 Header Encoding

Since the header has a set size, it does not follow the encoding structure set in Section 5.2.2. Each field is encoded using only its binary value.

6.3. Packet Types

The packet type indicates what kind of information is contained in the payload. The types are split into three categories: System, Account, and Chat.

6.3.1. System

These packets are used as generic responses to requests and are prefixed within this document with SYS.

[SYS_Success] Confirmation that the server has received and processed the client request successfully. The server responds with the packet type of the request packet.

[SYS_Error] Confirmation that the server has received the client request, but could not process it. Contains an error code and message string explaining the error.

6.3.2. Account

Packet types regarding account services are prefixed within this document with ACC.

[ACC_Login] A client request to login to the server chat program. Contains the user's username and password

[ACC_Login_Success] The server response to ACC_Login. Contains the user's id.

[ACC_Logout] A client request to log the user out of the server chat program. Contains the user's username and password

[ACC_Create] A client request to create an account for the server chat program. Contains the user's username and password

6.3.3. Chat

Once the user is connected, this is the main packet type used for chat services.

[CHT_Send] Contains a timestamp, the contents of the message, and the sender's username. The server will relay and broadcast this message to all connected users.

6.3.4. Enumerated values

Key (Decimal)	Key (Hex)	Value	Version Released
0	00	SYS_Success	1
1	01	SYS_Error	1
10	0A	ACC_Login	1
11	0B	ACC_Login_Success	1
12	0C	ACC_Logout	2
13	0D	ACC_Create	1
20	14	CHT_Send	2

6.4. Payload

6.4.1. Payload Structure

```
SYS_Success ::= SEQUENCE {
     packet_type ENUMERATED (1 BYTE)
}

SYS_Error ::= SEQUENCE {
     code    ENUMERATED (1 BYTE),
     message UTF8String
}

ACC_Login ::= SEQUENCE {
     username UTF8String,
     password UTF8String
}

ACC_Login_Success ::= SEQUENCE {
     id INTEGER (2 BYTES)
}

ACC_Logout ::= SEQUENCE {
     username UTF8String,
     password UTF8String
}

ACC_Create ::= SEQUENCE {
     username UTF8String,
     password UTF8String
}

CHT_Send ::= SEQUENCE {
     timestamp GeneralizedTime,
     content   UTF8String,
     username  UTF8String
}
```

6.4.2. Error Codes

These error codes are used in the “code” field of the Sys_Error payload above. The “message” field in the Sys_Error payload is the what’s written in the “Value” column in the table below.

Code (Decimal)	Code (Hex)	Value	Category
11	0B	Invalid User ID	Invalid Client Input
12	0C	Invalid Authentication	Invalid Client Input
13	0D	User Already Exists	Invalid Client Input
21	15	Generic Failure	Server Failure
31	1F	Invalid Request	Validation
32	20	Request Timeout	Validation

6.5. Packet Transmission Pairings

The following client request packets should expect the following server responses.

Client Request	Server Response
ACC_Login	ACC_Login_Success or SYS_Error
ACC_Logout	No Response
ACC_Create	SYS_Success | SYS_Error
CHT_Send	Broadcast CHT_Send

7. Server Management Protocol

7.1 Overview

The Server Management protocol is used to communicate between the server starter and server manager. Each packet is split into a header and a payload. Server starters can connect to a server manager, although only one server starter will be allowed to connect at a time. The primary purpose of this protocol is to either start or stop a server starter's associated chat server. Additionally, it details a way for the chat server to send diagnostic data about the current state of the server and its clients.

7.2 Header

The header holds information about the message type, version and payload.

7.2.1 Header Structure

```
ServerManagerMessage ::= SEQUENCE {
     messageType    PacketType (1 Byte),
     version        INTEGER (1 Byte),
     payloadLength  INTEGER (2 Bytes)
}
```

[message_type] indicates to the receiver what kind of information is being sent
[version] the protocol version that the sending program is using (currently supported: 3)
[payload_len] total length of the payload in bytes.

7.2.2 Header Encoding

Since the header has a set size, it does not follow the encoding structure set in Section 5.2.2. Each field is encoded using only its binary value.

7.3 Packet Types

The packet type indicates what kind of information is contained in the payload. The types are split into three categories: Manager Response, Server Diagnostic, and Server Commands.

7.3.1 Manager Response

These packets are used as generic responses to requests and are prefixed within this document with MAN. In this release, they are unused and can be ignored.

7.3.2 Server Diagnostics

Packet types regarding server status or usage information.

[SVR_Diagnostic] The server must intermittently send diagnostic or usage information to the server. This includes the number of active clients and messages sent during the server's lifetime.

[SVR_Online] The server must respond to a SVR_Start or SVR_Stop message with its status, if the chat server is online it will send this message.

[SVR_Offline] The server must respond to a SVR_Start or SVR_Stop message with its status, if the chat server is offline it will send this message.

7.3.3 Server Commands

Packet types used by the server manager to issue commands to the server starter.

[SVR_Start] Request to start the chat server.
[SVR_Stop] Request to stop the chat server.

7.3.4 Enumerated Values

Key (Decimal)	Key (Hex)	Value	Category	Version Released
0 - 9			Manager Response	
10	0A	SVR_Diagnostic	Server Diagnostics	1
11	0B	[Unused]*	Server Diagnostics	None
12	0C	SVR_Online	Server Diagnostics	3
13	0D	SVR_Offline	Server Diagnostics	3
14-19				
20	14	SVR_Start	Server Commands	2
21	15	SVR_Stop	Server Commands	2

*Unused value from prerelease version

7.4 Payload

7.4.1 Payload Structures

```
ServerDiagnostics ::= SEQUENCE {
     userCountOnline          INTEGER (2 Bytes),
     messagePerSession        INTEGER (4 Bytes)
}

SuccessResponse ::= SEQUENCE {
     respondingTo   PacketType
}

ServerOnline ::= SEQUENCE {
     ClientPort     UTF8STRING
}
```

8. Client Connection Protocol

8.1 Overview

The Client Connection protocol is used to connect clients to a server manager and request information about which server to connect to and employ the Chat Message protocol with. Clients will initially not know the IP address or port number of the server that is currently online, and must use this protocol to retrieve the IP and port number of the current running server, if one is online.

Clients initiate communication by sending a Client_GetIp packet detailed in section 8.2. The contents of this message are detailed in Section 8.2. Upon receiving the message, the server manager will respond with a MAN_ReturnIp packet detail in section 8.2. The contents of this message are detailed in Section 8.2.1.

8.2 Message Format

The Client Connection protocol must follow the following binary format:

```
ClientConnectionMessage ::= HEADER {
     messageType         ConnectionPacketType,
     version             INTEGER (1 Byte),
     serverOnline        BOOLEAN OPTIONAL,
     activeServerIp      UTF8STRING OPTIONAL,
     activeServerPort    UTF8STRING OPTIONAL
}
```

8.2.1 Contents of Message

[message_type] indicates to the receiver what kind of information is being sent
[version] the protocol version that the sending program is using (currently supported: 3)
[serverOnline] Indicates if there is an active chat server or not
[activeServerIp] The active chat server’s IP to connect to as a client
[activeServerPort] The active chat server’s port to connect to as a client

8.3 ConnectionPacketType Values

The ConnectionPacketType value is sent in binary format (see the Key (Hex) column for exact value in hexadecimal), and can only be one of the following values.

Key (Decimal)	Key (Hexadecimal)	Value	Category
0	00	CLIENT_GetIp	Client
1	01	MAN_ReturnIp	Server Manager

[CLIENT_GetIp Requests] The version number must be the same as the value noted in section 5.3 and must be sent as a single byte in integer format.
Only the messageType and version field are sent by the client, the serverOnline, activeServerIp and activeServerPort fields are not sent at all.

[MAN_ReturnIp Requests] The version number must be the same as the value noted in section 5.3 and must be sent as a single byte in integer format.
Upon receiving a CLIENT_GetIp request, the server manager must respond with a serverOnline value of 0 if no servers are online, or 1 if there is a server online. Additionally, if there is no server online the activeServerIp and activeServerPort should be set to 2 bytes of 0 each. If there is a server online the activeServerIp and activeServerPort must be set to a string representation of the IP number and the port number.

9. References

[BCP14] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997,

<http://www.rfc-editor.org/info/bcp14>.

[X680] ITU-T, "Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[X690] ITU-T, "Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015, <https://www.itu.int/rec/T-REC-X.690 >.

Contributors

Author’s Addresses

Blaise Klein
RFC Editor

bklein13@my.bcit.ca

Oceaan Pendharkar
RFC Editor

opendharkar@my.bcit.ca

Kevin Van Nguyen
kvnbanunu@gmail.com