Chat Message Protocol v3

This section details the protocol format used by clients and servers to manage user accounts, login, logout, and send chat messages.

Table of Contents

Version 3 Updates

- Protocol Version is program-specific, suggested implementation is a global version number that is checked upon receiving a request/response 0 < Received <= Supported
- Packet types in this documentation will now label the version # of its release
- Chat sending and receiving has been slightly edited:
    - The server should receive a chat request.
    - One big chat room will be used in this version.
    - Server should broadcast to all connected clients when a new message is received.
    - The server no longer sends a SYS_Success message to the sending client.
    - The clients will not send any confirmation messages to the server upon chat message receipt.

- Server manager no longer requests user count from server - servers will send user count to server manager at intervals set by each server team

Guidelines

- Header section is a fixed 6 bytes, not following TAG LEN CONTENT
- The payload follows BER (TAG LEN CONTENT)
- Every LEN is restricted to **1 BYTE**
- Data is represented in HEX values, but should be encoded as unsigned integer bytes
- 0 is a reserved User ID. **Servers do not assign a user ID 0.** ID 0 can be used by a user that does not know its ID on a server yet, or by the server responding in success or error cases.
    - Upon successful login, the server will respond with the user's assigned ID
- Each packet type/request should be single responsibility, do not bundle requests
    - Assume client and server do not share context
    - ie. if server receives a login request, do not also perform an acc_create action
- GeneralizedTime needs to be in YYYYMMDDhhmmssZ and stored as UTC.
- Protocol Version is program-specific, suggested implementation is a global version number that is checked upon receiving a request/response 0 < Received < Supported
- Packet types include the version # of its release to handle backwards compatibility

MESSAGE HEADER (6 Bytes)

TCP

- Packet type: 1 byte
- Protocol Version (Current: 2) (1 byte)
- Sender ID (2 bytes)
- Payload length (2 bytes)

Header Structure

```
Message ::= Sequence {
     packet_type    ENUMERATED (1 BYTE)
     version        INTEGER (1 BYTE)
     sender_id      INTEGER (2 BYTES)
     payload_len    INTEGER (2 BYTES)
     payload        CHOICE{
          SYS_Success | SYS_Error |
          ACC_Login | ACC_Login_Success | ACC_Logout |
          ACC_Create | ACC_Edit |
          CHT_send | LST_get | LST_response
     }
}
```

PAYLOAD

System Response (Sending the error type (types will be available in the clients))

```
SYS_Success ::= SEQUENCE {
     --sends back the packet type it is responding to
     packetType ENUMERATED (1 BYTE) {see packet type enum table},
} example

--Currently only includes client errors needed for Milestone 1
--but this will include any packet that the server could send
--back to indicate an error
SYS_Error ::= SEQUENCE {
     code      ENUMERATED (1 BYTE) see Error Codes,
     message   UTF8String
} example
```

Account

```
--The ACC_login is to get into an account that already exists
--If this account doesn't exist yet, the server should return an error
ACC_Login ::= SEQUENCE {
     username UTF8String,
     password UTF8String
} example

ACC_Login_Success ::= SEQUENCE {
     id INTEGER (2 BYTES),//client needs to know this ID now
} example

ACC_Logout ::= SEQUENCE {
     --No payload
} example

--ACC_Create is for an account that doesn't exist yet
--If it does already exist the server should return an error
ACC_Create ::= SEQUENCE {
     username UTF8String,
     password UTF8String
} example

ACC_Edit ::= SEQUENCE {
     edit_field ENUMERATED (1 BYTE) {username (0), password(1)},
     edit_value UTF8String //holds the new username or password
} example
```

Chat

```
CHT_Send ::= SEQUENCE {
     timestamp GeneralizedTime,
     content   UTF8String,
     username  UTF8String --The name of the msg sender
} example
```

Userlist Query (Considerations for future versions)

```
LST_Get ::= SEQUENCE {
     group_id  INTEGER (1 BYTE)
     filter    ENUMERATED (1 BYTE) { no_filter (0), online(1) }
}
```

```
--When encoding, use SEQ LEN for the list and have the LEN represent
--the number of Users instead of the actual length of the list.
--Do not encode SEQ LEN for each user
--Encode each field of User as is
LST_Response ::= SEQUENCE {
     list ::= SEQUENCE OF User
}
```

Struct Types

```
User ::= SEQUENCE {
     id INTEGER,
     username UTF8String,
     status ENUMERATED (1 BYTE) {offline(0), online(1),busy (2)}
}
```

Enums

Packet Types:

Key (Decimal)	Key (Hex)	Value	Category	Version Released
0	00	SYS_Success	System	1
1	01	SYS_Error	System	1
2 - 9				
10	0A	ACC_Login	Account	1
11	0B	ACC_Login_Success	Account	1
12	0C	ACC_Logout	Account	2
13	0D	ACC_Create	Account	1
14	0E	ACC_Edit	Account	-
15 - 19				
20	14	CHT_Send	Chat	2
22 - 29				
30	1E	LST_Get	Userlist	-
31	1F	LST_Response	Userlist	-

Error Codes

Error Code (decimal value)	Message
1X	Invalid Client Input
11	Invalid User ID
12	Invalid Authentication Information
13	User Already Exists

2X	Server Failure
21	Generic Server Failure*
3X	Validity Errors
31	Invalid request
32	Request timeout

*Placeholder for milestone 1

BER Encoding Scheme

Relevant BER Tags1

Tag (decimal)	Tag (hex)	Type
1	01	BOOLEAN
2	02	INTEGER
5	05	NULL
10	0A	ENUMERATED
12	0C	UTF8STRING
16 (48)*	10 (30)*	SEQUENCE/SEQUENCE OF
19	13	PrintableString
23	17	UTCTime
24	18	Generalized Time

*Always encoded as DEC(48)/HEX(30)

Tag-Length-Value Encoding

Each line sent using the BER encoding scheme has three parts.

- The tag designating the data type of the value (1 Byte)
- The length of the value (1 Byte)
- The data that fits within the length (1+ Bytes)

As an example, the line below denotes an integer with tag 1, of length 4, with a value of 12.
Hex: 02 04 00 00 00 0C
Binary: 00000010 00000100 00000000 00000000 00000000 00001100
Decimal: 2 4 12

Examples

Account Examples

Login Request | Client → Server

Line (HEX format)	Description
`0A`	Packet type set to ACC_login
`03`	Version set to 3
`00 00`	Sender ID set to 0 (User does not know their ID initially)
`00 16`	Payload length of 22
`0C 07 54 65 73 74 69 6E 67`	Username set to “Testing”
`0C 0B 50 61 73 73 77 6F 72 64 31 32 33`	Password set to “Password123”

Login Success Response | Server → Client

Line (HEX format)	Description
`0B`	Packet type set to ACC_Login_Success
`03`	Version set to 3
`00 00`	Sender ID set to 0 (Server Default)
`00 04`	Payload length of 4
`02 02 00 01`	Returning User ID as 1

Login Failure Response | Server → Client

Line (HEX format)	Description
`01`	Packet type set to SYS_Error
`03`	Version set to 3
`00 00`	Sender ID set to 0 (Server Default)
`00 27`	Payload length of 39

`02 01 0C`	Error code 12
`0C 22 49 6E 76 61 6C 69 64 20 41 75 74 68 65 6E 74 69 63 61 74 69 6F 6E 20 49 6E 66 6F 72 6D 61 74 69 6F 6E`	Error message “Invalid Authentication Information”

Logout Request | Client → Server

Line (HEX format)	Description
`0C`	Packet type set to ACC_logout
`03`	Version set to 3
`00 01`	Sender ID set to 1
`00 00`	Payload length of 0

Account Create Request | Client → Server

Line (HEX format)	Description
`0D`	Packet type set to ACC_Create
`03`	Version set to 3
`00 00`	Sender ID set to 0 (User does not know their ID initially)
`00 16`	Payload length of 22
`0C 07 54 65 73 74 69 6E 67`	Username set to “Testing”
`0C 0B 50 61 73 73 77 6F 72 64 31 32 33`	Password set to “Password123”

Account Create Success Response | Server → Client

Line (HEX format)	Description
`00`	Packet type set to SYS_Success
`03`	Version set to 3
`00 00`	Sender ID set to 0 (Server default)

`00 03`	Payload length of 3
`0A 01 0D`	Responding to ACC_Create message type: type: enum, length: 1 byte, value: 10

Account Edit Request | Client → Server

Line (HEX format)	Description
`0E`	Packet type set to ACC_Edit
`03`	Version set to 3
`00 01`	Sender ID set to 1
`00 09`	Payload length of 9
`0A 01 00`	edit_field set to 0: username
`0C 04 54 65 73 74`	edit_value: “Test” // new username

Account Edit Success Response | Server → Client

Line (HEX format)	Description
`00`	Packet type set to SYS_Success
`03`	Version set to 3
`00 00`	Sender ID set to 0 (Server default)
`00 03`	Payload length of 3
`0A 01 0E`	Responding to ACC_Edit message type: type: enum, length: 1 byte, value: 14

Chat Examples

Chat Send Request | Client → Server

Line (HEX format)	Description
`14`	Packet type set to CHT_Send
`03`	Version set to 3
`00 01`	Sender ID set to 1

`00 1E`	Payload length of 30
`18 0F 32 30 32 34 30 33 30 31 31 32 33 30 34 35 5A`	Timestamp set to 20240301123045Z
`0C 02 68 69`	Content set to “hi”
`0C 07 54 65 73 74 69 6E 67`	Username set to “Testing”

Chat Send Broadcast | Server → All Clients

Line (HEX format)	Description
`14`	Packet type set to CHT_Send
`03`	Version set to 3
`00 01`	Sender ID set to 1 // ID of the sender this message belongs to
`00 1E`	Payload length of 30
`18 0F 32 30 32 34 30 33 30 31 31 32 33 30 34 35 5A`	Timestamp set to 20240301123045Z 2024, March 1, 12:30:45 UTC
`0C 02 68 69`	Content set to “hi”
`0C 07 54 65 73 74 69 6E 67`	Username set to “Testing”

*** The Client does not send acknowledgement back

Server Management Protocol

This details the protocol format used for a server and server manager to communicate diagnostic data, or manage server state.

```
The messages between client and server manager must follow this
format:

ServerManagerMessage ::= SEQUENCE {
     messageType    PacketType,
     version        INTEGER,
     payloadLength  INTEGER,
     payload        CHOICE{
          UserCount | ServerDiagnostics |
          SuccessResponse | ErrorResponse |
          ServerOnline
     }
}

UserCount ::= SEQUENCE {
     userCount      INTEGER
}

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

SuccessResponse ::= SEQUENCE {
     respondingTo   PacketType
}

ErrorResponse ::= SEQUENCE {
     error          ErrorCode
     message        ErrorMessage
}

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

Enums

Packet Types:

Key (Decimal)	Key (Hex)	Value	Category	Version Released
0	00	MAN_Success	Manager	1

			Response	
1	01	MAN_Error	Manager Response	1
2 - 9				
10	0A	SVR_Diagnostic	Server Diagnostics	1
11	0B	USR_Online	Server Diagnostics	-
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
22-29				

BER Encoding Scheme

Relevant BER Tags

Tag (decimal)	Tag (hex)	Type
1	01	BOOLEAN
2	02	INTEGER
5	05	NULL
10	0A	ENUMERATED

12	0C	UTF8STRING
16	10/30*	SEQUENCE/SEQUENCE OF
19	13	PrintableString
23	17	UTCTime
24	18	Generalized Time

*Always encoded as 30

Tag-Length-Value Encoding

Each line sent using the BER encoding scheme has three parts.

- The tag designating the data type of the value (1 Byte)
- The length of the value (1 Byte)
- The data that fits within the length (1+ Bytes)

***The messageType and version** are 1 byte in length and do not follow BER encoding.
***The payloadLength** is 2 bytes in length and does not follow BER encoding

As an example, the line below denotes an integer with tag 1, of length 4, with a value of 12.
Hex: 02 04 00 00 00 0C
Binary: 00000010 00000100 00000000 00000000 00000000 00001100
Decimal: 2 4 12

Examples

Send Server User Count

Line (HEX format)	Description
`0A`	`SVR_Diagnostic` packet type
`03`	Version 3 of the protocol
`00 0A`	Payload Length of 10 bytes
`02 02 00 0A`	User Count of 10 reported by server
`02 04 00 00 00 64`	Message Count of 100 reported by server

Server Response in Success

Line (HEX format)	Description
00	MAN_Success packet type
03	Version 3 of the protocol
00 03	Payload Length of 3 bytes
0A 01 0A	Responding to USR_Count packet type

Server Request Start

Line (HEX format)	Description
14	SVR_Start packet type
03	Version 3 of the protocol
00 00	Payload Length of 0 bytes

Server Request Stop

Line (HEX format)	Description
15	SVR_Stop packet type
03	Version 3 of the protocol
00 00	Payload Length of 0 bytes

Server Responded Online

Line (HEX format)	Description
0C	SVR_ONLINE packet type
03	Version 3 of the protocol
00 00	Payload Length of 0 bytes

Server Responded Offline

Line (HEX format)	Description
0D	SVR_Offline packet type
03	Version 3 of the protocol

`00 00`	Payload Length of 0 bytes

Client Connection Protocol

This details a simple protocol used by clients and the server manager to initiate connection to the currently available server, if one is available.

The messages between client and server manager must follow this format:

```
ConnectionMessage ::= SEQUENCE{
     messageType          ConnectionPacketType,
     version              INTEGER,
     serverOnline         BOOLEAN,
     activeServerIp       UTF8STRING OPTIONAL,
     activeServerPort     UTF8STRING OPTIONAL
}
```

Enums

Packet Types:

Key (Decimal)	Key (Hex)	Value	Category	Version Released
0	00	CLIENT_GetIp	Client	1
1	01	MAN_ReturnIp	Server Manager	1

BER Encoding Scheme

Relevant BER Tags

Tag (decimal)	Tag (hex)	Type

1	01	BOOLEAN
2	02	INTEGER
10	0A	ENUMERATED
12	0C	UTF8STRING
16	10/30*	SEQUENCE/SEQUENCE OF

*Always encoded as 30

Tag-Length-Value Encoding

Each line sent using the BER encoding scheme has three parts.

- The tag designating the data type of the value (1 Byte)
- The length of the value (1 Byte)
- The data that fits within the length (1+ Bytes)

`*The messageType, version and serverOnline values do not follow BER, they are always only 1 byte each.`

Examples

Client requests the current active server's IP

Line (HEX format)	Description
`00`	Set the packet type to CLIENT_GetIp
`03`	Version 3 of the protocol

Server responds to client with active server IP

Line (HEX format)	Description
`01`	Set the packet type to `MAN_ReturnIp`
`03`	Version 3 of the protocol
`01`	There is an active server
`0C 0C 31 39 32 2E 31 36 38 2E 30 2E 31`	IP of server to connect 192.168.0.1
`0C 04 38 30 38 30`	Port of server to connect 8080

Server responds to client, no active servers

Line (HEX format)	Description
`01`	Set the packet type to `MAN_ReturnIp`
`03`	Version 3 of the protocol
`00`	There is no active server
`0C 00`	The IP for the server to connect to is empty
`0C 00`	The port of the server to connect to is empty