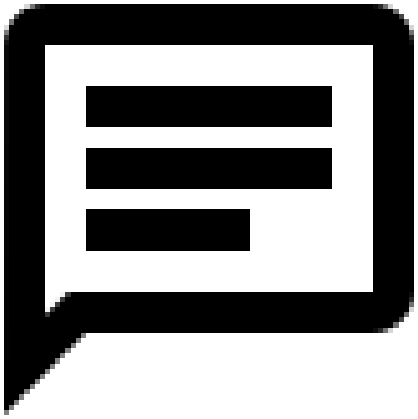
Chat Protocol v1.1



Prepared By:

Jordan Godau Giwoun Bae

Prepared For:

Data Communications & Internet-working

Computer Systems Technology, BCIT

Table of Contents

1.0 Executive Summary

The Chat protocol was designed for transmitting messages between mutually exclusive client based applications over a network. This document shall describe the particular details of the protocol including the formats, restrictions, usage examples, and all other information.

In general, the CPT Protocol has been designed with the basic responsibilities of most application layer protocols:

- Writing data from the network
- Receiving data from other network elements
- Defining an interface for server/client operations
- Error handling between multiple network elements

While the current protocol has been designed to support text-based messages, minor support has been added for other data objects such as imagery and audio. In its current state, it is strongly recommended that this protocol be used exclusively for text-based messages.

It should also be mentioned that the current protocol is intended to be used on top of the connection-oriented TCP protocol. This decision has been made to simplify the transmission of messages between multiple users across the network. When considering TCP, distinguishing between multiple origins is simple and packets are guaranteed to arrive at the destination. Additionally, a lost packet is easy to detect and can be simply re-transmitted if needed.

2.0 Protocol Limitations

In its current state, it is recommended that the CPT Protocol be used for transferring of simple ascii alpha-numeric messages. While simple support has been added for transmission of larger and more complex data, limited consideration has been carried out regarding the downstream effect of these implementations. Thus, it is important to clarify that any implementation of data other than text-based messages may cause CPT packet transmission to behave unexpectedly.

These cautionary recommendations should persist until more research has been completed by the authors to determine what types of support/semantics should be used to handle more complicated or advanced features in the protocol.

3.0 Terminology

The following section describes certain terminology where rigorous definitions are required with respect to The Protocol.

3.1 Global Channel

The Protocol defines the **Global Channel** as the server-side data structure from which all CHANNELs are derived from. The **Global Channel** shall be unique from all other channels in that:

- A LOGOUT message shall be the only packet capable of removing a user from the Global Channel
- LEAVE_CHANNEL packets shall never remove a user from the Global Channel.
- LOGIN should always place the user inside the Global Channel.
- All CHANNEL structures created should be a subset of the Global Channel
- The Global channel shall always be assigned a channel ID of 0 (zero)

Moreover, the Global Channel should contain the necessary information to identify client network elements in a typical client/server infrastructure. Additionally, sufficient information must exist in the data structure to store, define, or generate the following attributes:

- Username (optional)
- ID (**required**)

3.2 Channel

The Protocol defines a **Channel** as any data structure created as a subset of the Global Channel. The protocol places a special emphasis on the following properties of a **Channel**:

- A channel shall contain one or more users.
- User-based chat communication should only occur within Channels
- Channel IDs shall range from 1 to 65536 (exclusive)
- No Channel other than the Global Channel shall possess a Channel ID of 0 (zero).

The Channel shall be the backbone of user-based chat communication between client-elements in the CPT ecosystem. In essence, this means that whenever a user-based chat message is transmitted, it should be transmitted to all other members of the same channel, and should not be receivable by any other channel.

4.0 Protocol Packet Formats

The following section describes the various components of the Chat protocol. This includes the protocol message format, client/server specific behavior, restrictions, constraints, and limitations. Additionally, examples shall be included to demonstrate and reinforce the intended use of the protocol.

4.1 Client Packet Header Format

The following tables depict the protocol message header for Chat, along with a description of it's individual fields:

CPT Header Block	
VER	**CMD_CODE**
CHAN_ID	**MSG_LEN**
MSG	

FIELD	DESCRIPTION	SIZE (BITS)
VER	Version	8
CMD_CODE	Used for providing instructions to the server.	8
CHAN_ID	Mainly for responses, this field will show what channel the message belongs in	16
MSG_LEN	The number of bytes in the messages	16
MSG	The message being transmitted	8*[0~65536)
	Total:	48 + (8*[0~65536))

4.2 Client Packet Header Fields

The following Sections describe the protocol fields in greater details. This includes the range of specified values, as well as examples to demonstrate the intended use of the protocol message.

4.2.1 VER Field

The VER field is reserved for providing the version of the protocol of a message being sent. 8 bits are reserved for the VER field and the field is designed to store up to 256 different versions of the chat protocol.

All the contents, headers, and commands must adhere to the protocol of the version sent, which includes the size allocated for message headers. In the case of the server receiving an outdated (deprecated) message, the outdated message will be dropped and the server shall respond back with the minimum version it’s expecting.

4.2.2 CMD Field

The CMD field is designated for providing instructions to a chat-based server. These instructions shall be sent from the client to the server, and shall never be altered by the server once the message has been transmitted from the client. It should be mentioned that the proposed CMD options shall be prescriptive in their instruction,

however the implementation is up to the software designer. In essence, the *semantics* of the CMD shall be followed, however the exact way these instructions are carried out should be done in a way that best fits the software architecture.

The 8 bits comprising the CMD field shall be utilized to support up to 256 commands. Each CMD, its point code values, as well as a description of its intended use is presented in the table below:

4.2.3 CHAN_ID Field

The CHAN_ID field is reserved for indicating a unique identifier for specifying the channel ID in which the message is being sent in.

This field consists of 16 bits representing the unique identifier of the channel ID. This allows a maximum of 65536 channels that can be uniquely distinguished, including the Global Channel.

In addition, this field will be used by the server to identify the channel in which the message belongs to. Therefore, chat client responses with the same CHAN_ID shall be set to be in the same channel and will be able to communicate with one another.

For the implementation of public and private channels, access flags will be used. For example, access flags with a value of 0 will be reserved for public channels and access flags with a value of 1 will be reserved for private channels.

4.2.4 MSG_LEN Field

The MSG_LEN field of the protocol shall define the length of the message intended for transmission (contained in the MSG field). The value in this field shall be an 16-bit sequence representing a number of bytes in the message field.

Unless there is a valid reason to do so, the default value of this field shall be zero. In other words, if no bytes are transmitted in the MSG field of the message, the value of this field shall be zero.

4.2.5 MSG Field (SEND packet)

The MSG field shall contain a message that the client application intends to SEND. Any messages transmitted from the client shall be destined to all users residing in the server-side CHANNEL_ID specified in the CPT SEND packet. The message shall be composed of any sequence of bytes, with the exception that the message size shall not exceed 65535 bytes.

Furthermore, the bytes contained within the content field shall have no restriction on the type of content being transmitted.

Additionally, some restrictions are provided as a suggestion regarding the MSG field and its relationship with other fields in the packet header:

1. **A message containing content size of 0 bytes should contain a MSG_LEN value of 0**
2. **The MSG field should not be read if the CHAN_ID has not been specified.**

4.2.6 MSG Field (LOGIN packet)

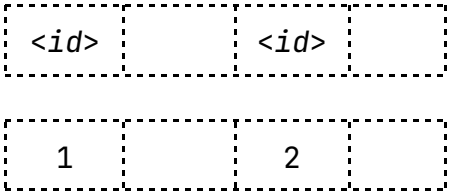
In the special case of sending a LOGIN packet, the MSG field may be optionally utilized to pass a client username inside the MSG field of the packet.

No restrictions are placed on usernames, with the exception that **the username shall not exceed 12 bytes in length (12 Bytes in size).**

4.2.7 MSG Field (CREATE_CHANNEL packet)

In the special case of sending a CREATE_CHANNEL packet, the MSG field may be optionally used to specify a list of server-side user IDs. These IDs shall be used to instruct the server to add the IDs to the newly created CHANNEL.

Any time a list of USER IDs is provided, the contents of the MSG field **shall strictly adhere to the following format in the following examples**:



An empty cell represents **whitespace**

4.3 Server Packet Header Format

The following tables depict the protocol message header for Chat, along with a description of it's individual fields:

CPT Response Header Block		
RES_CODE	**MSG_LEN**	**MSG**

FIELD	DESCRIPTION	SIZE (BITS)
RES_CODE	Server Code	8
MSG_LEN	The number of bytes in the messages	16
MSG	The message being transmitted	8*[0~65536)
	Total:	48 + (8*[0~65536))

4.4 Server Packet Header Fields

The following section describes details regarding the Server Response Packet header fields, including restrictions, uses, and properties.

4.4.1 RES_CODE

The server will send the client an appropriate RES_CODE based on the client message. For example, a RES_CODE value of 1 corresponds to a successful transmission and operation with respect to the message sent to the server. Additional RES_CODEs are provided to signal different events processed by the server. Refer to the Chat Protocol API for a full list of RES_CODES.

4.4.2 MSG_LENx

Refer to section 3.2.4.

4.4.3 MSG

The message (MSG) field shall include optional messages from server to client. Depending on which RES_CODE the Server Packet Header responds with, the MSG field shall include the sub packets.
Please refer to section 4.5 for sub packets for Special cases.

4.5 Response Packet Contents (Special Cases)

The following section describes special response sub-packets. The sub-packets described are included as a rigorous definition of data from any server implementing the CPT Protocol.

4.5.1 MSG Response Sub-Packet

The MSG response sub-packet is the recommended output format for messages relayed by the server between different client elements. This format is not prescriptive, however, failure to comply with this format may result in unusual results in the client-side data stream.

Message Response Sub-Packet		
CHANNEL_ID	**USER_ID**	**MSG_LEN**
MSG		

FIELD	DESCRIPTION	SIZE (BITS)
CHANNEL_ID	Channel id of the channel the message was sent in	16
USER_ID	Id of the user that sent the message to the channel	16
MSG_LEN	The number of bytes in the messages	16
MSG	The message being transmitted	8*[0~65535)

Total:	48 + (8*[0~65535))

4.5.2 USERNAME_ID_PAIR format

The USERNAME_ID_PAIR is a prescriptive complimentary format for the USER_LIST sub-packet. Any server response containing either one or more records of user information shall contain this information, exactly as described below.

USERNAME_ID_PAIR	
USER_ID	**USERNAME**

FIELD	DESCRIPTION	SIZE (BITS)
USER_ID	The client id	16
USERNAME	USERNAME 12 bytes allocated	8*12
	Total:	16 + 8*12

4.5.3 USER_LIST Response Sub-Packet

The USER_LIST sub-packet is a special response format reserved exclusively for server responses to a client GET_USER packet. This sub-packet is composed of newline-separated USERNAME_ID_PAIRs **derived from the server-side data structure used to represent the CHANNEL_ID in the received GET_USER packet.**

Message Response Sub-Packet	
NUM_USERS	**USER_LIST**

FIELD	DESCRIPTION	SIZE (BITS)
NUM_USERS	The number of users in the list	16

USER_LIST	List of USERNAME_ID_PAIR	8*14*[0~1000)
	Total:	16 + 8*14*[0~1000)

APPENDIX A. Client to Server Commands

CMD	VALUE (Hex)	VALUE (dec)	Description
SEND	0x01	1	Shall initiate transmission of a message to the specified <CHANNEL_ID>.
LOGOUT	0x02	2	Shall disconnect the client from the server, removing any instance of user identification in the server.
GET_USERS	0x03	3	Shall be used to obtain a list of users available to receive a transmitted MSG. If channel is 0 then all users are returned, if channel is given then only users in a given channel are returned.
CREATE_CHANNEL	0x04	4	Shall be used to instruct a server application to create a data structure (a channel) used for transmitting messages to multiple client network elements.
JOIN_CHANNEL	0x05	5	Shall be used to add a client element to an existing data structure (a channel) intended for transmitting messages to multiple client network elements.
LEAVE_CHANNEL	0x06	6	Shall be used to remove a single client network element from an existing multi-client data structure (a channel).
LOGIN	0x07	7	Shall be used to create a private channel, only visible to the members who comprise it.
	…	…	…
RESERVED	0xFF	255	Reserved for future CMDs

APPENDIX B. Server Response Table

CMD	CMD_CODE (HEX)	CMD_CODE (decimal)	DESCRIPTION
SEND	0x01	1	Designated SEND CMD code
LOGOUT	0x02	2	Designated LOGOUT CMD code
GET_USERS	0x03	3	Designated GET_USERS CMD code
CREATE_CHANNEL	0x04	4	Designated CREATE_CHANNEL CMD code
JOIN_CHANNEL	0x05	5	Designated JOIN_CHANNEL CMD code
LEAVE_CHANNEL	0x06	6	Designated LEAVE_CHANNEL CMD code
LOGIN	0x07	7	Designated LOGIN CMD code
CREATE_VCHAN	0x08	8	Designated CREATE VOICE CHANNEL code
MESSAGE	0x09	9	The channel id is in the CHAN_ID, msg contents are a message sub-packet
USER_CONNECTED	0x0A	10	The ID of the connected user, msg contents are the username
USER_DISCONNECTED	0x0B	11	The ID of the disconnected user, msg contents are the username
MESSAGE_FAILED	0x0B	11	Message could not be delivered, could be followed by a USER_DISCONNECTED
CHANNEL_CREATED	0x0C	12	Response sent to requestee, if the channel is created it provides the channel ID in the <CHAN_ID> section
CHANNEL_CREATION_ERROR	0x0D	13	Response sent to requestee, if the channel cannot be created

CHANNEL_DESTROYED	0x0E	14	Response sent to all members of channel when it is destroyed
USER_JOINED_CHANNEL	0x0F	15	Response sent to all members when a new client joins the channel
USER_LEFT_CHANNEL	0x10	16	Response sent to all members when a member leaves the channel
USER_LIST	0x11	17	A response message from GET_USERS
UKNOWN_CMD	0x12	18	Unknown command error
LOGIN_FAIL	0x13	19	Login failed error
UKNOWN_CHANNEL	0x14	20	Unknown channel ID error
BAD_VERSION	0x16	21	Bad CPT Version number error
SEND_FAILED	0x17	22	Message failed to send
CHAN_ID_OVERFLOW	0x18	23	Channel ID exceeds limit
MSG_OVERFLOW	0x19	24	Message size too big
MSG_LEN_OVERFLOW	0x1A	25	Message length too big
CHAN_EMPTY	0x1B	26	No users in channel
INVALID_ID	0x1C	27	Given ID is invalid for any reason
FAILURE	0x1D	28	Operation failed
UNAUTH_ACCESS	0x1E	29	Access to resources is forbidden
SERVER_FULL	0x1F	30	Server at maximum capacity
…	…	…	…
RESERVED	0xFF	255	Reserved for future CMDs

APPENDIX C. Example Packets

Note: The cpt_packet examples below have commas to separate each field for viewing purposes. The actual packet will not have any space or character between each field.

CLIENT TO SERVER:

Example 1: Sending a message to channel 0 with message

Packet	VER	CMD_CODE	CHAN_ID	MSG_LEN	MSG
	1	1	0	17	hello this is bob
cpt_packet	0x01,0x00,0x0000,0x0011,hello this is bob\n				

Example 2: Sending a packet for joining a channel 5

Packet	VER	CMD_CODE	CHAN_ID	MSG_LEN	MSG
	1	5	5	0	-
cpt_packet	0x01,0x06,0x0005,0x0000				

Example 3: Requesting the user list in channel 8

Packet	VER	CMD_CODE	CHAN_ID	MSG_LEN	MSG
	1	3	8	0	-
cpt_packet	0x01,0x03,0x0008,0x0000				

Example 4: Requesting the user list in all channels

Packet	VER	CMD_CODE	CHAN_ID	MSG_LEN	MSG
	1	3	0	0	-

cpt_packet	0x01,0x03,0x0000,0x0000

Example 5: Creating a channel for yourself

Packet	VER	CMD_CODE	CHAN_ID	MSG_LEN	MSG
	1	4	NULL	NULL	-
cpt_packet	0x01,0x04,0x0000,0x0000, -				

Example 6: Creating a channel for yourself and users 4, 5 and 6 in it

Packet	VER	CMD_CODE	CHAN_ID	MSG_LEN	MSG
	1	4	NULL	6	“4 5 6 “
cpt_packet	0x01,0x04,0x0000,0x0000,”4 5 6 “				

SERVER TO CLIENT:

Note: The cpt_packet examples below have commas to separate each field for viewing purposes. The actual packet will not have any space or character between each field.

Example 1: Transmitting a message to a member of channel 0 from user 6.

Example	CPT Header Block		Message Response
	RES_CODE	MSG_LEN	DATA
	1	18	hello this is bob
cpt_packet	0x01,0x0018,hello this is bob\n		

Example 2: Transmitting a list of users from a group 4.

```
Current users in group 4:
     USER_ID: 55          USERNAME: GB1234567899
     USER_ID: 66          USERNAME: HB1234567899
     USER_ID: 2           USERNAME: BB1234567899
```

Example	CPT Header Block		Message Response Sub-Packet		
	RES_CODE	MSG_LEN	NUM_USERS	USER_LIST (USERNAME_ID_PAIR)	
				USER_ID	USERNAME
	10	45	3	55	GB1234567899
				66	HB1234567899
				2	BB1234567899
cpt_packet	0x0A,0x002D,0x03,0x0037,GB1234567899,0x0042,HB1234567899,0x0002,BB1234567899				

APPENDIX D: ABNF

```
VER_ID      = 8*8(BIT)
CMD_CODE    = 8*8(BIT)
CHAN_ID     = 16*16(BIT)
USER_ID     = 16*16(BIT)
MSG_LEN     = 16*16(BIT)
MSG         = *65535(CHAR)
USERNAME    = 12*12(CHAR)

CLIENT_TO_SERVER_PACKET = VER_ID CMD_CODE CHAN_ID MSG_LEN [MSG]

MSG_SUBPACKET = CHAN_ID USER_ID MSG_LEN [MSG]

USERNAME_ID_PAIR = USER_ID USERNAME

USERLIST_SUBPACKET = MSG_LEN 0*1000(USERNAME_ID_PAIR)

USER_ID_STRING = 1*5(DIGIT)(space_character)

CREATE_CHANNEL_SUBPACKET = 0*1000[USER_ID_STRING]

SERVER_TO_CLIENT_PACKET = VER_ID CMD_CODE MSG_LEN [SUBPACKET]
```