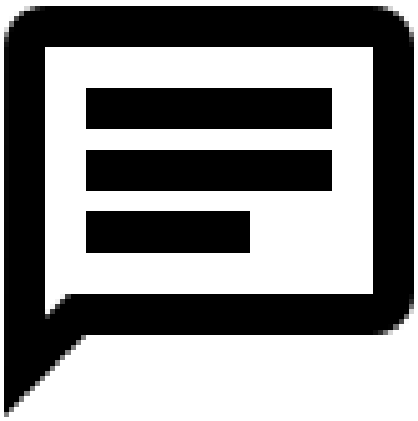
CPT API (Version 2.1)



Prepared By:

Jordan Godau Giwoun Bae

Mar 25, 2022

Table of Contents

CPT Enums

commands		
Name	**Value**	**Description**
SEND	1	Designated SEND CMD code
LOGOUT	2	Designated LOGOUT CMD code
GET_USERS	3	Designated GET_USERS CMD code
CREATE_CHANNEL	4	Designated CREATE_CHANNEL CMD code
JOIN_CHANNEL	5	Designated JOIN_CHANNEL CMD code
LEAVE_CHANNEL	6	Designated LEAVE_CHANNEL CMD code
LOGIN	7	Designated LOGIN CMD code
CREATE_VCHAN	8	Designated CREATE_VCHAN CMD code

CPT Structs

CptRequest		
Name	**Type**	**Description**
version	uint8_t	CPT version number.
command	uint8_t	CPT command code.
channel_id	uint16_t	CPT destination channel ID.
msg_len	uint16_t	CPT message length in bytes.
msg	char *	CPT message contents.

CptResponse		
Name	**Type**	**Description**
code	uint8_t	CPT response code
data_size	uint16_t	Size of struct member <data> in bytes.
data	uint8_t *	CPT response data

CPT Server Response Codes

CPT Server Response Codes			
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
…	…	…	…
RESERVED	0xFF	255	Reserved for future CMDs

CPT Request Packet Builder Functions

cpt_request_init

```
/**
 * Initialize CptRequest object.
 *
 * Dynamically allocates a cpt struct and
 * initializes all fields.
 *
 * @return Pointer to cpt struct.
*/
CptRequest * cpt_request_init();
```

cpt_request_destroy

```
/**
 * Free all memory and set fields to null.
 *
 * @param cpt   Pointer to a cpt structure.
*/
void cpt_request_destroy(CptRequest * cpt);
```

cpt_request_reset

```
/**
 * Reset packet parameters.
 *
 * Reset the packet parameters,
 * and free memory for certain params.
 *
 * @param packet    A CptRequest struct.
*/
void cpt_request_reset(CptRequest * packet);
```

CPT Response Packet Builder Functions

cpt_response_init

```
/**
 * Initialize CptResponse server-side packet.
 *
 * Initializes a CptResponse, returning a dynamically
 * allocated pointer to a CptResponse struct.
 *
 * @param res_code    Received client-side packet.
 * @return Pointer to a CptResponse object.
 */
CptResponse * cpt_response_init();
```

cpt_response_destroy

```
/**
 * Destroy CptResponse object.
 *
 * Destroys CptResponse object, freeing any allocated memory
 * and setting all pointers to null.
 *
 * @param response  Pointer to a CptResponse object.
 */
void cpt_response_destroy(CptResponse * response);
```

cpt_response_reset

```
/**
 * Reset packet parameters.
 *
 * Reset the response parameters, and free memory for certain params.
 *
 * @param response      Pointer to a CptResponse object.
*/
void cpt_response_reset(CptResponse * response);
```

CPT Request Functions

cpt_login

```
/**
 * Prepare a LOGIN request packet for the server.
 *
 * Prepares a LOGIN request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to instruct the server to
 * persist the client's information until cpt_logout() is called.
 *
 * @param cpt            CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @param name           Client login name.
 * @return The size of the serialized packet in <serial_buf>.
*/
size_t cpt_login(void * client_info, uint8_t * serial_buf, char * name);
```

cpt_logout

```
/**
 * Prepare a LOGOUT request packet for the server.
 *
 * Prepares a LOGOUT request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to instruct the server to remove
 * any instance of the requesting client's information.
 *
 * @param cpt            CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @return Size of the resulting serialized packet in <serial_buf>
*/
size_t cpt_logout(void * client_info, uint8_t * serial_buf);
```

cpt_get_users

```
/**
 * Prepare a GET_USERS request packet for the server.
 *
 * Prepares a GET_USERS request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to instruct the server to
 * send back a list of users specified by the <channel_id>.
 *
 * @param client_info    CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @param channel_id     The ID of the CHANNEL to get users from.
 * @return Size of the resulting serialized packet in <serial_buf>
*/
size_t cpt_get_users(void * client_info, uint8_t * serial_buf, uint16_t
channel_id);
```

cpt_create_channel

```
/**
 * Prepare a CREATE_CHANNEL request packet for the server.
 *
 * Prepares a CREATE_CHANNEL request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to instruct the server to create
 * a new channel.
 *
 *      > <user_list> may be optionally passed as user selection
 *        parameters for the new Channel.
 *      > If <members> is not NULL, it will be assigned to the
 *        MSG field of the packet.
 *
 * @param cpt            CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @param user_list      Whitespace separated user IDs as a string.
 * @return Size of the resulting serialized packet in <serial_buf>
*/
size_t cpt_create_channel(void * client_info, uint8_t * serial_buf, char *
user_list);
```

cpt_join_channel

```
/**
 * Prepare a JOIN_CHANNEL request packet for the server.
 *
 * Prepares a JOIN_CHANNEL request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to instruct the server to add
 * the client's information to an existing channel.
 *
 * @param client_info    CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @param channel_id     The target channel id.
 * @return Size of the resulting serialized packet in <serial_buf>
*/
size_t cpt_join_channel(void * client_info, uint8_t * serial_buf, uint16_t
channel_id);
```

cpt_leave_channel

```
/**
 * Prepare a LEAVE_CHANNEL request packet for the server.
 *
 * Prepares a LEAVE_CHANNEL request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to remove the client's information
 * from the channel specified by <channel_id>.
 *
 * @param client_info    CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @param channel_id     The target channel id.
 * @return Size of the resulting serialized packet in <serial_buf>
*/
size_t cpt_leave_channel(void * client_info, uint8_t * serial_buf, uint16_t
channel_id);
```

cpt_send

```
/**
 * Prepare a SEND request packet for the server.
 *
 * Prepares a SEND request to the server. If successful,
 * the resulting data in <serial_buf> will contain a CPT packet
 * with the necessary information to instruct the server to SEND
 * the message specified by <msg> to every user in the channel
 * specified within the packet CHAN_ID field.
 *
 * @param client_info    CPT packet information and any other necessary data.
 * @param serial_buf     A buffer intended for storing the result.
 * @param msg            Intended chat message.
 * @return Size of the resulting serialized packet in <serial_buf>.
*/
int cpt_send(void * client_info, uint8_t * serial_buf, char * msg);
```

cpt_create_vchannel

```
/**
* Prepare a CREATE_VCHAN request packet for the server.
*
* Prepares a CREATE_VCHAN request to the server. If successful,
* the resulting data in <serial_buf> will contain a CPT packet
* with the necessary information to instruct the server to create
* a new voice channel.
*
*  <user_list> may be optionally passed as user selection
*  parameters for the new voice channel.
*
* @param cpt            CPT packet information and any other necessary data.
* @param serial_buf     A buffer intended for storing the result.
* @param user_list      Whitespace separated user IDs as a string.
* @return Size of the resulting serialized packet in <serial_buf>
*/
size_t cpt_create_vchannel(void * client_info, uint8_t * serial_buf, char *
user_list);
```

CPT Response Functions

cpt_login_response

```
/**
 * Handle a received 'LOGIN' protocol message.
 *
 * Use information in the CptPacket to handle
 * a LOGIN protocol message from a connected client.
 *
 * If successful, the protocol request will be fulfilled,
 * updating any necessary information contained within
 * <server_info>.
 *
 * @param server_info   Server data structures and information.
 * @param name          Name of user in received Packet MSG field.
 * @return Status Code (SUCCESS if successful, other if failure).
 */
int cpt_login_response(void * server_info, char * name);
```

cpt_logout_response

```
/**
 * Handle a received 'LOGOUT' protocol message.
 *
 * Uses information in a received CptRequest to handle
 * a LOGOUT protocol message from a connected client.
 *
 * If successful, will remove any instance of the user
 * specified by the user <id> from the GlobalChannel
 * and any other relevant data structures.
 *
 * @param server_info   Server data structures and information.
 * @return Status Code (SUCCESS if successful, other if failure).
 */
int cpt_logout_response(void * server_info);
```

cpt_get_users_response

```
/**
 * Handle a received 'LOGOUT' protocol message.
 *
 * Uses information in a received CptRequest to handle
 * a GET_USERS protocol message from a connected client.
 *
 * If successful, the function should collect user information
 * from the channel in the CHAN_ID field of the request packet
 * in the following format:
 *
 *  <user_id><whitespace><username><newline>
 *
 * Example given:
 *      1 'Clark Kent'
 *      2 'Bruce Wayne'
 *      3 'Fakey McFakerson'
 *
 * @param server_info   Server data structures and information.
 * @param channel_id    Target channel ID.
 * @return Status Code (SUCCESS if successful, other if failure).
 */
int cpt_get_users_response(void * server_info, uint16_t channel_id);
```

cpt_join_channel_response

```
/**
 * Handle a received 'JOIN_CHANNEL' protocol message.
 *
 * Uses information in a received CptRequest to handle
 * a JOIN_CHANNEL protocol message from a connected client.
 * If successful, function should add the requesting client
 * user into the channel specified by the CHANNEL_ID field
 * in the CptPacket <channel_id>.
 *
 * @param server_info   Server data structures and information.
 * @param channel_id    Target channel ID.
 * @return Status Code (SUCCESS if successful, other if failure).
 */
int cpt_join_channel_response(void * server_info, uint16_t channel_id);
```

cpt_create_channel_response

```
/**
 * Handle a received 'CREATE_CHANNEL' protocol message.
 *
 * Uses information in a received CptRequest to handle
 * a CREATE_CHANNEL protocol message from a connected client.
 *
 * If a <user_list> was received in the MSG field of the packet,
 * function will also parse the <user_list> string and attempt
 * to add the requested user IDs to the channel.
 *
 * If <id_list> is NULL, function will create a new channel with
 * only the requesting user within it.
 *
 * @param server_info   Server data structures and information.
 * @param id_list       ID list from MSG field of received CPT packet.
 * @return Status Code (SUCCESS if successful, other if failure).
 */
int cpt_create_channel_response(void * server_info, char * id_list);
```

cpt_leave_channel_response

```
/**
 * Handle a received 'LEAVE_CHANNEL' protocol message.
 *
 * Use information in the CptPacket to handle
 * a LEAVE_CHANNEL protocol message from a connected client.
 * If successful, will remove any instance of the user
 * specified by the user <id> from the GlobalChannel
 * and any other relevant data structures.
 *
 * @param server_info   Server data structures and information.
 * @param channel_id    Target channel ID.
 * @return Status Code (SUCCESS if successful, other if failure).
 */
int cpt_leave_channel_response(void * server_info, uint16_t channel_id);
```

cpt_send_response

```
/**
 * Handle a received 'SEND' protocol message.
 *
 * Uses information in a received CptRequest to handle
 * a SEND protocol message from a connected client.
 *
 * If successful, function will send the message in the
 * MSG field of the received packet to every user in the
 * CHAN_ID field of the received packet.
 *
 * @param server_info   Server data structures and information.
 * @param name          Name of user in received Packet MSG field.
 * @return Status Code (0 if successful, other if failure).
 */
int cpt_send_response(void * server_info, char * name);
```

cpt_create_vchannel_response

```
/**
* Handle a received 'CREATE_VCHAN' protocol message.
*
* Uses information in a received CptRequest to handle
* a CREATE_VCHAN protocol message from a connected client.
*
* If a <user_list> was received in the MSG field of the packet,
* function will also parse the <user_list> string and attempt
* to add the requested user IDs to the channel.
*
* If <id_list> is NULL, function will create a new voice channel with
* only the requesting user within it.
*
* @param server_info   Server data structures and information.
* @param id_list       ID list from MSG field of received CPT packet.
* @return Status Code (SUCCESS if successful, other if failure).
*/
int cpt_create_vchannel_response(void * server_info, char * id_list);
```

CPT Serialize Functions

cpt_serialize_request

```
/**
* Serialize a CptRequest struct for transmission.
*
* @param cpt    A CptRequest struct.
* @return       Size of the serialized packet.
*/
size_t cpt_serialize_request(CptRequest * req, uint8_t * buffer);
```

cpt_serialize_response

```
/**
* Serialize a CptResponse object for transmission.
*
* @param cpt    A CptResponse object.
* @return       Size of the serialized packet.
*/
size_t cpt_serialize_response(CptResponse * res, uint8_t * buffer);
```

CPT Parse Functions

cpt_parse_response

```
/**
 * @brief Parse serialized server response.
 *
 * @param response  Address to a CptResponse object.
 * @param buffer    Serialized response from server.
 * @return Pointer to filled CptResponse.
 */
CptResponse * cpt_parse_response(uint8_t * res_buf, size_t data_size);
```

cpt_parse_request

```
/**
* Create a cpt struct from a cpt packet.
*
* @param packet    A serialized cpt protocol message.
* @return A pointer to a cpt struct.
*/
CptRequest * cpt_parse_request(uint8_t * req_buf, size_t req_size);
```