

CPT Chat Server/Client User Guide

Jordan Godau

Giwoun Bae

April 11, 2022

Purpose	3
Installing	3
Obtaining	3
Building	3
Installing	3
Running	3
Server	3
Client	3
Features	3
Example	4
Running the server	4
Start up	5
Running the client & logging in	5
Showing the menu	6
Entering Commands	6
Getting Active User Information	7
Creating channel	7
Join Channel	7
Leave Channel	8
Logout	8
Sending Messages	9

Purpose

GiJo_chat is a server-client program that supports:

- Send messages and receive messages in the same channel
- Implementing CPT chat protocol v 1.1 + v 2.0 for voice support for clients.

Installing

Obtaining

<https://github.com/JordanGunn/GiJo-chat-server.git>

Building

```
mkdir GiJo-chat-server/cmake-build-debug
cmake -S GiJo-chat-server -B
      GiJo-chat-server/cmake-build-debug
cmake --build GiJo-chat-server/cmake-build-debug/
```

The compiler can be specified by passing one of the following to cmake:

- `-DCMAKE_C_COMPILER="gcc" -DCMAKE_CXX_COMPILER="g++"`
- `-DCMAKE_C_COMPILER="clang" -DCMAKE_CXX_COMPILER="clang++"`

Installing

```
sudo cmake --install /cmake-build-debug
```

Running

Server

```
cpt_chat_server
```

Client

```
Cpt_chat_client --LOGIN username --host 127.0.0.1 --port 8080
cpt_messenger
```

Features

Chat program that allows users to communicate through use of CPT protocol.

User facing program utilises a command line interface with the following features:

- Creation of channels by specification of User ID
- Retrieving users by name & ID from the server
- Joining of Channels by specifying the Channel ID
- Leaving of channels
- Login & Logout
- Storing & Updating of useful User information.
- Sending of messages
- Sending/ Receiving voice packets over UDP protocol
- Recording/ Playing voice on Linux

Example

Running the server

The server is currently set to run on the PORT and IP specified in the tcp_server_config.h file. Further abstraction is needed in the future - with possible solution being a simple server.config file.

Once an IP or Port has been set, the executable can be called from the command line using:

```
./cpt_chat_server
```

See below:

```
#define IP_LOCAL_INET      "192.168.1.106"
#define IP_LOCAL_LB        "127.0.0.1"
#define IP_LOCAL_BC        "192.168.1.255"
#define PORT_8080          "8080"
#define PORT_8888          "8888"
#define PORT_TO_HELL       "666"
#define LISTEN_BACKLOG_3   3
#define LISTEN_BACKLOG_5   5
#define LISTEN_BACKLOG_7   7
#define LISTEN_BACKLOG_32  32
#define BASE_10            10
```

Start up

Running the client & logging in

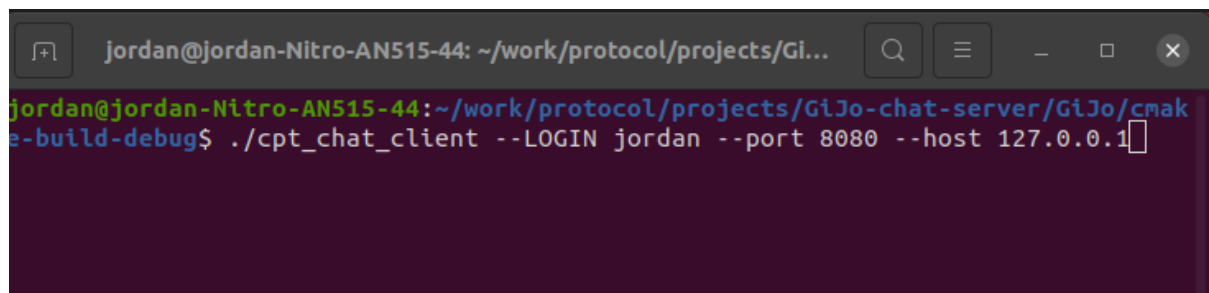
If the server is running, the client can connect. If the server is not running, the LOGIN state will not be set, and the client will fail to connect immediately.

A client may login by executing the program with the following arguments:

```
--LOGIN <user_name>
--port <port_number>
--host <host_name> (or IP address)
```

Example given:

```
./cpt_chat_client --LOGIN jordan --port 8080 --host 127.0.0.1
```

A terminal window screenshot showing a user named 'jordan' on a machine named 'jordan-Nitro-AN515-44'. The user is in the directory '~/work/protocol/projects/GiJo-chat-server/GiJo/cmake-build-debug'. The command being executed is './cpt_chat_client --LOGIN jordan --port 8080 --host 127.0.0.1'. The terminal has a dark purple background and a light blue prompt. The window title bar shows the file path and standard window controls (search, menu, zoom, close).

```
jordan@jordan-Nitro-AN515-44: ~/work/protocol/projects/GiJo-chat-server/GiJo/cmake-build-debug$ ./cpt_chat_client --LOGIN jordan --port 8080 --host 127.0.0.1
```

If a user starts the program and logs in successfully, they will receive a confirmation and find themselves at a prompt, which displays the username, and the user's current channel number:

```
jordan@jordan-Nitro-AN515-44:~/work/protocol/projects/GiJo-chat-server/GiJo/cmake-build-debug$ ./cpt_chat_client --LOGIN jordan --port 8080 --host 127.0.0.1

Logged in to chat server as jordan@127.0.0.1:8080

[ jordan | (channel 0) ] $
```

Showing the menu

From the prompt, the user can access any of their options by typing @menu :

```
[ giwoun | (channel 0) ] $ @menu
=====
Choose from the following options...

[0] send <message>
[1] get-users <chan_id>
[2] create-channel "<uid-1> <uid-2>.. <uid-n>"
[3] join-channel <chan_id>
[4] leave-channel <chan_id>
[5] logout <name>
[6] menu
=====
```

Entering Commands

Commands allow the user to perform a number of actions to organise and direct their messages to particular users who are also logged in.

All commands distinguish themselves from regular messages by beginning with an @ symbol. The commands available are:

@get-users	Get available users from the server.
@create-channel	Create a channel for users to communicate.
@join-channel	Join a channel by ID number.
@leave-channel	Leave a channel by ID number
@join-vchannel	Join a voice channel by ID number
@logout	Logout of the chat program, and close the program.
@menu	Show the menu, with a list of available commands.

Getting Active User Information

Get available users from the server. If no argument is specified, the command will get users from the channel number specified at the user prompt.

Example Given with no argument (gets users from channel 0):

```
[ giwoun | (channel 0) ] $ @get-users
ID: 4   NAME: giwoun
ID: 0   NAME: ROOT_USER
```

```
[ giwoun | (channel 1) ] $ @get-users 1
ID: 4   NAME: giwoun
ID: 5   NAME: jordan
```

Creating channel

Create a channel. If no argument is specified, a channel will be created with only the requesting user. If a user specifies a USER LIST, the server will *attempt* to find all users in the USER LIST and add them to the channel.

The USER LIST can be specified by space-separated IDs wrapped in quotation marks.

Example Given:

```
@create-channel "3 4 5"
```

See below:

```
[ giwoun | (channel 0) ] $ @create-channel "5 4"
Successfully created channel with users: [ 5 4 ]
```

Join Channel

A channel can be joined by specifying the CHANNEL ID after the command. If the channel does not exist, the server will respond with ERROR CODE 17:

See below:

```
[ jordan | (channel 0) ] $ @join-channel 1
[ jordan | (channel 1) ] $ @get-users
ID: 4   NAME: giwoun
ID: 5   NAME: jordan

[ jordan | (channel 1) ] $ █
```

Leave Channel

A channel can be left by using the `@leave-channel` command. A channel can be left even if the user is currently in another channel. The server will attempt to remove the requesting user from the channel, and if successful, the user will no longer be visible from the channel to other users within that channel.

In the following example, two users are part of CHANNEL 1. The user **giwoun** leaves the channel, and when **jordan** runs the `@get-users` command, **giwoun** is no longer visible. See below:

```
[ giwoun | (channel 1) ] $ @leave-channel 1
Successfully left channel 1
[ giwoun | (channel 0) ] $ █
```

```
[ jordan | (channel 1) ] $ @get-users
ID: 5   NAME: jordan
```

Logout

Running this command will remove all instances of the user's information from the server and shut-down the program. It is important to not that regardless of the server's success in

logging the user out, the user program will shut-down immediately, thus making sure that a user can log out regardless of the server's condition.

See below:

```
[ giwoun | (channel 0) ] $ @logout
jordan@jordan-Nitro-AN515-44:~/work/protocol/projects/GiJo-chat-server/GiJo/cmake-build-debug$
```

Sending Messages

Sending a message involves text entered at the prompt that does not start with an @ symbol.

Messages will be output to the CPT Messenger. At this time, the CPT Messenger must be started individually from the client program. However, even if starting the messenger after running the client, sent and received messages will still be immediately displayed on startup.

See below:

```
[ jordan | (channel 0) ] $ Hey there!
[ jordan | (channel 0) ] $ Anyone there?
[ jordan | (channel 0) ] $

jordan@jordan-Nitro-AN515-44: ~/work/protocol/projects/GiJo-chat-server/GiJo/cmake-build-debug$ ./cpt_messenger
Hey there!

Anyone there?


```