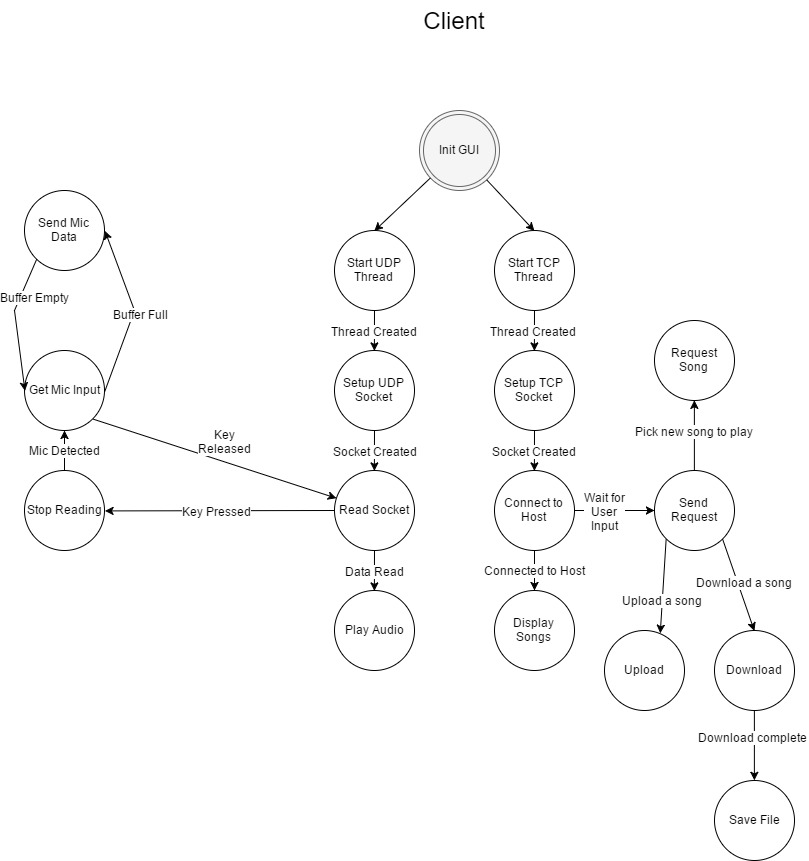
# Comm Audio – Design Doc

## Client FSM



## Client Pseudo Code

**INIT GUI**

1. Add File menu item
   1. Add exit sub menu item
2. Add About menu item
3. Add Station Group box
   1. Add IP text box
   2. Add Port text box
   3. Add Connect button
4. Add Clients group box
   1. Add Client list display box
5. Add Track group box
   1. Add Current track playing text
6. Add Tracklist group box
   1. Add Update button
   2. Add Upload button
   3. Add track list display box
   4. Add Play Selected Song button
   5. Add Download Selected Song button

**START UDP THREAD**

1. Create a new thread off of the main one
2. No information needs to be given to the thread
3. Function should be one that is responsible for streaming audio and sending mic input

**SETUP UDP SOCKET**

1. Initialize versioning (ERROR CHECK)
2. Create UDP socket (ERROR CHECK)
3. Set broadcast option (ERROR CHECK)
4. Initialize address information
5. Bind socket (ERROR CHECK)

**READ SOCKET**

1. Prepare receive buffer
2. Set buffer as a queue of audio sources (stream)
3. Start Play Audio Thread with receive buffer
4. In a forever loop
   1. If a Mic signal is received (mic key pressed)
      1. suspend Play audio thread
      2. Go to stop reading State
   2. Receive from Multicast IP
   3. Store packet content into buffer
   4. Update Stream Queue

**STOP READING (these checks are happening to the read socket state)**

1. Set stop reading flag
2. Check for stop reading flag
   1. If the flag has been set
      1. Wait for key release event

**GET MIC INPUT**

1. Open a wave input stream (ERROR CHECK)
2. Prepare a record buffer
3. Add buffer to record to
4. Start recording (write to buffer) (ERROR CHECK)
5. If buffer full
   1. Go to send mic data state
6. If key released
   1. Stop recording
   2. Go to read socket state to read incoming data from server

**SEND MIC DATA**

1. Compress buffer data
2. Write buffer to socket (ERROR CHECK)

**PLAY AUDIO**

This state exists to actually play the audio we've received. Due to the need of updating our buffer constantly, it requires its own thread.

1. Wait for initial buffer filling.
2. While buffer has content
   1. Play stream buffer

**START TCP THREAD**

1. Should be run off the main thread (no need to explicitly create a new one)
2. Purpose is to manage the connection to the server and send requests

**SETUP TCP SOCKET**

1. Init versioning (ERROR CHECK)
2. Create TCP socket (ERROR CHECK)

**CONNECT TO HOST**

1. Extract IP address from the GUI
2. Check to see that the IP is a valid one
3. If IP is invalid
   1. Exit CONNECT TO HOST*[STATE]*
4. Initialize address information
5. Bind the socket to the address
6. Connect to the server
7. Go to DISPLAY SONGS*[STATE]*

**DISPLAY SONGS**

1. Read on the socket and specify a completion routine
   1. Make sure a message was received without error
   2. Parse the message and store each delimited song name into a list of songs
2. Update GUI with the song list

**SEND REQUEST**

1. Get user request
   1. If request is to play a song
      1. Go to REQUEST SONG*[STATE]*
   2. If request is to Upload a song
      1. Go to UPLOAD*[STATE]*
   3. If request is to download a song
      1. Go to DOWNLOAD*[STATE]*
   4. If request is to close connection
      1. Send teardown Request over TCP
      2. Exit SEND REQUEST*[STATE]*

**REQUEST SONG**

1. Extract selected song from song list.
2. Send a message containing
   1. The type “request song”
   2. The song name
3. Wait for response (Error or success)
4. Display response

**UPLOAD**

1. Prompt user to select file to upload
2. If file is not in the correct format (.wav)
   1. Display error message
   2. Exit UPLOAD*[STATE]*
3. Prepare buffer of file sections
4. Send request type and file name
5. Wait for response (Error or continue)
   1. If error
      1. Display message
      2. Free buffer
      3. Exit UPLOAD*[STATE]*
6. For each section in buffer
   1. Send section of data
7. Send transfer complete message
8. Free buffer

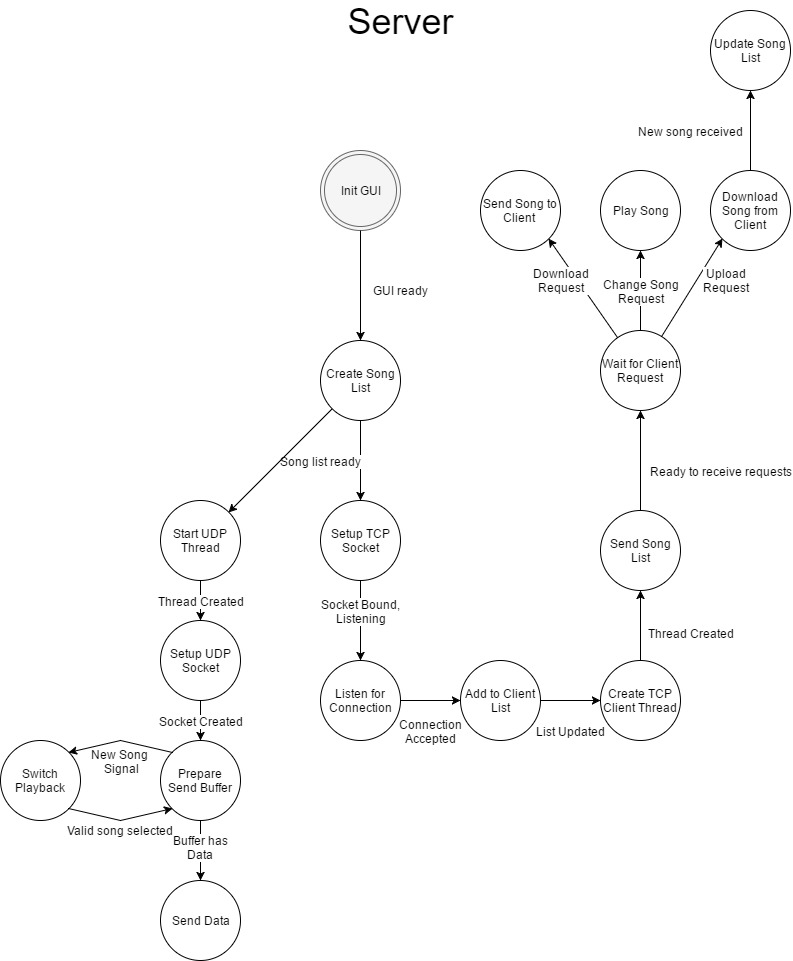
**DOWNLOAD**

1. Extract selected song from song list.
2. Send Request over TCP
3. Receive a message and specify a completion routine
   1. If message is request complete
      1. Close file
      2. Exit DOWNLOAD*[STATE]*
   2. If message is Song not found
      1. close file
      2. delete file
      3. Exit DOWNLOAD*[STATE]*
   3. Write Packet data to file
   4. Post another receive with the same completion routine

**SAVE FILE**

1. Open a Qt file browser
2. Get the save path
3. Open the file for writing only
4. Store the data into the file
5. Close the file

## Server FSM



## Server Pseudo Code

**INIT GUI**

1. Add File menu item
   1. Add exit sub menu item
2. Add About menu item
3. Add Server group box
   1. Add Start button
   2. Add Stop button
   3. Add status text
4. Add Client group box
   1. Add Client list display box
5. Add Song group box
   1. Add Current song playing
   2. Add Rewind button
   3. Add Play/Pause button
   4. Add Stop button
   5. Add Fast forward button
   6. Add Track list title text
   7. Add Shuffle radio button
   8. Add Track list display box
   9. Add Play selected track button

**CREATE SONG LIST**

1. Create a buffer to hold all the song names
2. Get the path containing the folder of songs
3. Check for valid directory
4. while there are files containing the extension ".wav" //Using boost::filesytem
   1. store the filename in the song name buffer

**START UDP THREAD**

1. Create a new thread off the main one
2. Give the multicast address to the thread
3. Purpose is to send audio streaming

**SETUP UDP SOCKET**

1. Initialize versioning (ERROR CHECK)
2. Setup multicast group address and port settings
3. Setup multicast intervals (how often to send)
4. Setup time to live (scope of multicast live)
5. Create UDP socket (ERROR CHECK)
6. Initialize address information
7. Bind socket (ERROR CHECK)
8. Join multicast group

**PREPARE SEND BUFFER**

1. As a loop:
   1. Open Song file (.WAV format) from song list at current song counter index.
   2. Prepare buffer
   3. Load initial chunks
   4. Go to SEND DATA*[STATE]*
      1. If return of Send Data is signal code
         1. Go to top of loop
   5. Close Song File
   6. Increment Song counter
   7. If song counter is greater than the number of songs available
      1. Reset song counter

**SEND DATA**

1. In a forever loop:
   1. Convert chunk to packet
   2. Send packet to multicast IP address
   3. Remove chunk from buffer
   4. Advance buffer forward
   5. Load next chunk into buffer
      1. If last chunk loaded was last chunk of song:
         1. break loop
2. Prepare/send remaining packets
3. Return how much song was played

**SWITCH PLAYBACK**

1. Send signal to stop sending audio packets
2. Find specified song in the song list
3. Open the media file
4. Send contents to PREPARE BUFFER[STATE]

**LISTEN FOR CONNECTION**

1. while the program is alive
   1. accept an incoming connection on an accepting socket
   2. go to ADD TO CLIENT LIST[STATE]

**ADD TO CLIENT LIST**

1. Extract the address from the accepting socket
2. Add the address to a client list buffer

**CREATE TCP CLIENT THREAD**

1. Create a new thread to service each TCP connected client
2. Give the thread the socket the connection was accepted on
3. Purpose of the thread is to handle requests from each client simultaneously

**SETUP TCP SOCKET**

1. Initialize versioning (ERROR CHECK)
2. Create TCP socket (ERROR CHECK)
3. Initialize address information
4. Bind the socket (ERROR CHECK)
5. Set socket to listen (ERROR CHECK)

**SEND SONG LIST**

1. Create message buffer
2. For every entry in the song list
   1. Append entry to the message
   2. Append New line character to message
3. Send message
4. Go to wait for client request State

**WAIT FOR CLIENT REQUEST**

1. While connection is open
   1. Poll connection for any TCP requests
   2. If a request is detected:
      1. Parse request type
         1. If request is to play song
            1. Go to PLAY SONG REQUEST*[STATE]* with song name
         2. If request is to download a song
            1. Go to SEND SONG TO CLIENT*[STATE]* with song name
         3. If request is to upload a song with desired filename
            1. Go to DOWNLOAD SONG FROM CLIENT*[STATE]*
         4. If request is to close connection
            1. Initiate teardown
2. End thread

**PLAY SONG REQUEST**

1. Check Song list for requested song
   1. If song exists on server
      1. Send Signal to streaming thread
      2. Set Song counter to appropriate index
      3. Send Request successful message
   2. If song Does not Exist
      1. Send “Song could not be found” message

**SEND SONG TO CLIENT**

1. Check Song list for requested song
   1. If song exists on server
      1. Prepare Buffer
      2. send each chunk in the buffer
      3. send request complete message
   2. If song doesn't exist
      1. Send Song could not be found message over TCP

**DOWNLOAD SONG FROM CLIENT**

1. Get name of uploaded song
2. Check existing songs.
3. If song already exists:
   1. Send Song Already exists message over TCP
   2. Exit DOWNLOAD SONG FROM CLIENT*[STATE]*
4. Prepare file for writing with song name
5. Send continue Message over TCP
6. Post a receive with a specified completion routine
   1. If transfer complete is indicated
      1. stop receiving
   2. Append data to file
   3. Post another receive with same completion routine
7. Close file
8. Go to Update Song list state with name of new song.

**UPDATE SONG LIST**

1. Read the filename that was passed in
2. add it to the song list buffer
3. Add new radio button to GUI with song name