COMP 4985
Comm Audio
Updated Design Document

Luke Lee	A00970469
Juliana French	A00998091
Alex Xia	A00991905
Vafa Dehghan Saei	A00983481

April 17, 2018

Table of Contents - Pseudocode

MainWindow

A class that controls all UI elements, including updating them with info when called to do so by another module

This is the entry point of the Windows menu-driven application. It is called by main, and creates the GUI based application, populates it with UI elements and enters an idle state waiting for user interaction

MainWindow constructor

```
{
    Initialize application windows
    Create application window
    Create reference to settings window
    Create menu items and set menu bar
        Transfer
        Connect
        Disconnect
        Settings
            Link clicking this to settings window
    Add buttons to windows
        Play
        Pause
        Stop
        Save Song
        Multicast
        Fast Forward
        Slow Down
    Add Progress bar slider //not a media player without one
        Add respective duration & progress display elements
    Add line edit for song selection
    Add Microphone / streaming buttons
        Start speaker
        Start microphone
        Stop microphone
    Disable all microphone buttons
    Set up table widget view with two columns
    Create reference to streaming module
    Start new background thread to run streaming module
        // Streaming & buffering will likely be CPU heavy
    Create reference to media player module
        Connect menu items and buttons to their action functions
}
```

UpdateSelectedFile

```
{
    Set column to 0;
    Get text at current selected row and column 0 as filename
    Set filename to song selection line edit
    Store filename to current setting
}
```

UpdatePlaylist

```
{
    // This function is called on the client side
    Clear current playlist
    If the updated playlist is not empty
        Parse the playlist into file names and file sizes
        Insert each file name and their size to table widget
    Else
        Display a message saying no audio file on server
}
```

AlertWrongFileType

```
{
    Create a popup message box
    Display a message saying must use .wav for streaming
}
```

EnableConnect

```
{
    Enable “Connect” menu item
    Disable “Disconnect” menu item
}
```

EnableDisconnect

```
{
    Enable “Disconnect” menu item
    Disable “Connect” menu item
}
```

UpdateSongProgress

```
{
    // call this function to update media player’s slider UI element according to actual song
    // progress
    Take 1 int argument, for seconds of song played
    Initialize local variable for minutes song played (seconds divided 60)
    Initialize local variable for seconds of song in minute to display(seconds mod 60)
    Set UI element to display song progress with time format “00:00”
    Move slider to current position of song
}
```

InitializeSongDuration

```
{
    // call this function at start of song so media player slider will accurate map to song
    // progress & duration
    Takes 1 int argument, for seconds of song played
    Initialize local variable for minutes song played (seconds divided 60)
    Initialize local variable for seconds of song in minute to display(seconds mod 60)
    Set UI element to display song duration with time format “mm:ss”
    Move slider to current position of song
    Set range of slider to song length in seconds
}
```

ToggleStreaming

```
{
    If ready for streaming
        Disable “Start Speaker” button
        If host type is client
            Enable “Start Mic” button
        Enable “Close Mic” button
    Else
        Enable “Start Speaker” button
        Disable “Start Mic” button
        Disable “Close Mic” button
}
```

MainWindow Destructor

```
{
    // must call this to end thread streaming module is on
    Join streaming module thread
    Delete settings reference
    Delete ui reference
    Exit program
}
```

UpdateSettings

```
{
    If host type is client
        Call OnActionClientTriggered
    Else if host type is server
        Call OnActionServerTriggered
    If transfer mode is file transfer
        Disable “Start Speaker” button
        Disable “Start Mic” button
        Disable “Close Mic” button
    Else
        Enable “Start Speaker” button
    If transfer mode is mulitcast and host type is server
        Enable “Multicast” button
    Else
        Disable “Multicast” button
}
```

LoadPlaylist

```
{
    //call this function as server
    Scan current project directory for wav files
        Parse found file names
        Add file names to UI element to display & to select
        Set file names and file sizes as one string to be sent to client
}
```

DisplayPlaylistByRow

```
{
    Create a table widget item
    Set item text with the given file name
    Disable editability of the item
    Insert item into current row and first column of table widget
}
```

DisplayFileSizeByRow

```
{
    Create a table widget item
    Set item text with the given file size
    Disable editability and selectability of the item
    Insert item into current row and second column of table widget
}
```

ClearPlaylist
```
{
    Clear table widget entries
    Clear current list of file names
    Clear current list of file sizes
}
```

OnActionConnectTriggered
```
{
    If host type is server
        Call LoadPlaylist and store playlist as a string
        Call Connect on transferer object
    Else if host type is client
        Call Connect on transferer object
}
```

OnActionDisconnectTriggered
```
{
    Call Disconnect on transferer reference
}
```

OnSaveButtonClicked
```
{
    // This function is called for file transfer on client side
    If host type is client
        If table widget has row number greater than zero
            Call SongSelected on transferer object with selected file name
}
```

OnActionSettingsTriggered
```
{
    Display a new popup settings window
}
```

UpdateReceiverStatus
```
{
    Update receiver status label with given text string
}
```

UpdateSenderStatus

```
{
    Update sender status label with given text string
}
```

ShowFilePicker

```
{
    Create an open file dialog to the current directory
    Set selected file name to song selection line edit
    Store filename to current setting
}
```

SettingsWindow

Not enough features to be a module. A popup UI class that doubles as storage of user-set settings of the modes to run program in. Consist of only string getter functions.

GetHostMode

```
{
    Returns either “Client” or “Server”
}
```

GetIpAddress

```
{
    Returns a IPv4 address, ie “127.0.0.1”
}
```

GetTransferMode

```
{
    Returns 1 of 3 modes: “file Transfer”, “microphone”, or “streaming”
}
```

GetFileName

```
{
    Returns file name of file to send via streaming as server
}
```

ToggleClientServerUi

```
{
    If host type is client
        Enable the IP address line edit on settings window
    Else if host type is server
        Disable the IP address line edit on settings window
}
```

SetFileName

```
{
    Set the file name to the selected file name from Main Window
}
```

MediaplayerModule

Module/class that controls playback of an audio file already saved in a local directory
Should be able to handle all audio file types, and have advanced features over playback
Should be just a wrapper for a high-level media player API, so the UI class MainWindow doesn't directly control audio.

MediaPlayerModule Constructor

```
{
    Instantiate a new QMediaPlayer object
    Set volume to 100%
}
```

MediaPlayerModule Destructor

```
{
    Delete the QMediaPlayer object
}
```

Play

```
{
    If playback stopped or unstarted
        Open file with fileName
        Set playback speed to 1x
    If filename is empty string
        Show warning as popup
        Return
    Call API play on file
}
```

Pause

```
{
    Wrapper function for API pause function that MainWindow can call
```

}

Stop

{
    Wrapper function for API stop function that MainWindow can call
}

FastForward

{
    Wrapper function for API function to increment play back rate by 0.1x that MainWindow can call
}

SlowForward

{
    Wrapper function for API function to decrement play back rate by 0.1x that MainWindow can call
}

ChangeSongPosition

{
    Wrapper function for API function to reassign song's current position to play from
    Takes a int in seconds to move to
}

IOSocketPair

A container class for a pair of socket/connections, used for streaming
Members:

- Sending (client) socket
- Receiving(server listening) socket
- Pointer to audio output stream, either speaker or null // server needs 1 / each client
- Pointer to audio input stream, either mic or audio file// server needs 1 / each client

IOSocketPair Constructor

{
    Instantiate a new QAudioInput object
    Instantiate a new QAudioOutput object
    Set output volumn to 1
    Initialize the recv socket to nullptr
    Initialize send socket as a new QTcpSocket

```
    Initialize a send stream as QDataStream with the send socket
}
```

IOSocketPair Destructor

```
{
    Stop the audio input
    Stop the audio output
    If send socket is not null
        Read all remaining data in the socket
        Disconnect the socket
        Set socket to null
    If recv socket is not null
        Read all remaining data in the socket
        Delete socket when program ends
    If send stream is not null
        Delete send stream
    Delete audio input
    Delete audio output
}
```

StreamingModule

StreamingModule should keep a (hash)map of IOSocketPair objects, mapped by the IP of their host:
A connection requires a client socket & a server listening socket.
For streaming without worrying about buffering issues, sockets should be read/write only to run at 100% capacity of connection
Each client will have its own server, which returns a single socket to do all the receiving. It'll have a socket to connect to a server
The server instance will have its own listening socket, which returns 1 or more client sockets that connected.

```
For each client socket returned:
    Add it to map of IOSocketPairs:
    Key: client ip address
    Value: IOSocketPair object
        sendSocket:to client
        recvSocket: from client
        Audio input stream:
            if mic mode, speaker
            if streaming mode, null
        Audio output stream:
            If mic mode, mic
            If streaming mode, file
```

```
Every time a socket disconnects:
    Get the disconnect socket's IP
    Find the IOSocketPair with the given IP in the map
    Stop the stream coming from the socket
    Remove the IOSocketPair from the map
```

StreamingModule Constructor

```
{
    Initialize a QAudioFormat object
    Set format sampling rate to 96000
    Set format channel count to 1
    Set sample size to 16
    Initialize a receiver as a new QTcpServer object
    Call ClientConnected when a new connection is received
}
```

StreamingModule Destructor

```
{
    Delete the receiver object
    Delete the audio format object
}
```

StartReceiver

```
{
    If receiver is already listening
        Return from function
    If transfer mode is streaming or multicast and host type is server
        If file type is not .wav
            Display warning message and return from function
    Set port number to server port (8000)
    If host type is client
        Set port number to client port (7000)
    Start receiver to listen for connection on server socket
}
```

AttemptStreamConnect

```
{
    // This function will only be called in client mode, server mode connects elsewhere
    If host type is client
        Try to connect to server using given ip & port 8000 (port for running server)
        Add the newly made socket to map of IOSocketPair objects
            On client this'll be the only send socket
}
```

AttemptStreamDisconnect

```
{
    // This function is called when user disconnects from UI
    If it's already disconnecting
        Return from function
    Set already disconnecting flag to true
    Increment through map of IOSocketPairs and remove & disconnect every socket in list
    Set main server socket to stop listening
}
```

GetSocketError

```
{
    Print socket error
}
```

RemoveSocketPair

```
{
    Get client socket to be removed from connection list
    Delete the specified client socket
    Remove the specified client socket from connection list
}
```

StartAudioInput

```
{
    Grab client from connection list
    If transfer mode is microphone
        Start client audio input from mic
    If transfer mode is streaming
        If host type is server
            Open the audio file to stream
            Read all bytes in file and put it to send stream to send to client
            Close the audio file
            Update sender status message
        If host type is client
            Update sender status message
    If transfer mode is multicast and host type is client
```

```
        Update sender status message
}
```

StartAudioOutput

```
{
    Grab client from connection list
    If the client is already null
        Return from function
    If client output state is idle or stopped
        Start the client audio output
}
```

MulticastAudioInput

```
{
    // This function gets called when "Multicast" button is clicked from Main Window
    If host type is server
        Update sender status message
        For each client in connection list
            Open the audio file to stream
            Read all bytes in file and put it to send stream to send to client
            Close the audio file
}
```

ClientConnected

```
{
    Set the recv socket to the next pending connection on server receiver
    If host type is client
        Find the client from the connection list
        Set client's socket to recv socket
    If host type is server
        Initialize a new IOSocketPair
        Set the new socket pair with received client address
        Insert received socket to connection list
}
```

ClientDisconnected

```
{
    Get the disconnecting client address from recv socket
    Call RemoveSocketPair on the recv socket
}
```

ServerDisconnected

```
{
    Update receiver status to let client knows server disconnects
}
```

TransferModule

Module/class that is responsible for file transfering mode for both server and client. When server connects in file transfer mode, it will load a playlist from the current directory and display on table widget view. When a client connects, it will receive the playlist from server and display in the table on client's side. Client can then request a file to download by sending the file name to the server.

TransferModule Constructor

```
{
    Set QTcpServer receiver to null
    Set QTcpSocket io socket to null
}
```

TransferModule Destructor

```
{
    Delete io socket
    Delete server receiver
}
```

Connect

```
{
    If host type is client
        If io socket is not null
            Initialize as a new QTcpSocket
        Call HandleConnect when io socket is connected
        Call HandleDisconnect when io socket is disconnected
        Call ClientReceivedBytes io socket is ready to read
        Connect the io socket with entered server IP address
    If host type is server
        Load playlist to send
        If receiver is null
            Initialize as a new QTcpServer
        Call ClientConnected when receiver receives new connection
        Start listening on server socket
}
```

Disconnect

```
{
    If io socket is not null
        Close the socket
    If receiver is not null
        Close the receiver socket
    Set transmitting flag to false
    Send a disconnected signal to main window
}
```

HandleConnect

```
{
    Send a connected signal to main window
}
```

HandleDisconnect

```
{
    Update receiver status message on main window
    If io socket is not null
        Close the socket
}
```

ClientReceivedBytes

```
{
    If still transmitting
        Open a QFile with given file name
        Open a QDataStream with the file
        Write received byte array to the stream
        Subtract expected bytes-to-recv from received bytes
        If bytes-to-recv is equal to 0
            Display a message showing file transfer complete
            Set transmitting flag to false
        Close file for writing
        Return from function
    Read first 8 bytes from io socket for descriptor bytes
    If descriptor is “filelist”
        Read all bytes from socket and store to playlist
    If descriptor is “filesize”
        Read all bytes from socket and parse file size as a number
    If descriptor is “filebyte”
        Set transmitting flag to true
        Open a QFile with given file name
        Open a QDataStream with the file
        Write received byte array to the stream
        Subtract expected bytes-to-recv from received bytes
        Close file for writing
}
```

SongSelected

```
{
    Append descriptor “filename” to the selected filename
    Open io socket with a QDataStream for writing
    Write filename string with descriptor to the socket
}
```

ServerReceivedBytes

```
{
    Read first 8 bytes from io socket for descriptor bytes
    If descriptor is “filename”
        Read all bytes from socket and store to filename
        Open a QFile with the received filename
        Get the file size and append with descriptor “filesize” in the front
        Write file size string to io socket stream
    If descriptor is “filesize”
        Read all bytes from socket and parse file size as a number
        If file size number received from client matches with actual file size
            Open the file for writing
            Read all bytes from the file and append descriptor “filebyte” in front
            Write file bytes to io socket stream
            Close the file
}
```

ClientConnected

```
{
    // This function is only called on the server side when a new client first connected
    Set io socket to server receiver’s next pending connection
    Load playlist from current directory as a single string
    Append the playlist string with descriptor “filelist” in front
    Write the playlist string to io socket stream
}
```