**Manual:**

To use “MusicPlayer”, the user must first launch the server.exe, which is allocated in “ChineseSpotify\x64\Debug\server.exe”.

After that, the client can use MusicClient.exe which is allocated in MusicPlayer\bin\Debug\MusicClient.exe. In the same folder, there is spdll.dll file that has all the needed functions and methods to provide functionality to MusicClient.exe.

To open the actual spdll.dll solution to modify the functions/methods, navigate to SP\_DLL folder and open the solution.

Please note that inside the ChineseSpotify solution, you will also find “client” project, which was my starting point. Ignore it, as it is not used anywhere.

**Also please note that you will have to change the directory of music files inside the server.cpp to appropriate directory on you machine.**

Below you will find the existing constructors/destructors, functions, etc. with explanation.

**DirectSoundManager.cpp:**

**This C++ code is part of an audio application that utilizes the DirectSound API, a part of the DirectX suite used for high-performance audio playback.**  
**Constructor**

The DirectSoundManager constructor initializes member variables and calls InitializeDirectSound(hwnd).

ZeroMemory is used to initialize the secondaryBuffers array.

**Destructor**

The destructor (~DirectSoundManager) releases resources to prevent memory leaks. It releases each secondary buffer in the secondaryBuffers array, the primary buffer, and the DirectSound object itself.

**Initialization Method**

InitializeDirectSound method is responsible for setting up DirectSound. It calls methods to create the DirectSound object, set the cooperative level, create the primary buffer, and create secondary buffers.

**DirectSound Object Creation**

CreateDirectSoundObject creates the DirectSound object. If DirectSoundCreate8 fails, it throws a runtime error.

**Setting Cooperative Level**

SetCooperativeLevel sets the cooperative level for the application. This determines how the application's use of DirectSound interacts with other applications. If setting the level fails, it throws a runtime error.

**Primary Buffer Creation**

CreatePrimaryBuffer creates the primary sound buffer. This is a DirectSound buffer that can be used to control global sound parameters but is not typically used for streaming audio. The function initializes a DSBUFFERDESC structure and calls CreateSoundBuffer. On failure, it throws an exception.

**Secondary Buffer Creation**

CreateSecondaryBuffers creates secondary buffers for audio streaming. These are the buffers where audio data will actually be loaded and played from.

It sets up a WAVEFORMATEX structure to describe the format of the audio, and then creates each buffer in the secondaryBuffers array. If creation fails, an exception is thrown.

**Get Secondary Buffer**

GetSecondaryBuffer retrieves a secondary buffer based on an index. This allows the rest of the application to access and use these buffers for audio playback. It throws an out\_of\_range exception if the index is invalid.

**AudioStreamer.cpp:**

**This C++ code is part of an audio streaming application, specifically handling the streaming of audio data from a server and playing it through DirectSound buffers.**

**Constructor**

Initializes the AudioStreamer with a network socket (SOCKET clientSocket) and two DirectSound secondary buffers (LPDIRECTSOUNDBUFFER secondaryBuffer1 and secondaryBuffer2).

Sets currentBufferIndex to 0, indicating which secondary buffer is currently in use.

**Destructor**

Ensures that the audio streaming thread is properly joined before the object is destroyed, preventing thread-related issues or crashes.

**Method: StartStreaming**

Starts audio streaming in a separate thread.

The thread executes the StreamAudioData method.

**Method: Pause**

Toggles playback pause status.

If pause is true, stops the current buffer. If false, resumes playback if there is data in the buffer.

**Method: SetVolume**

Sets the volume for audio streaming.

Converts a linear scale volume to a logarithmic decibel scale for DirectSound.

Applies this volume setting to both secondary buffers.

**Method: StreamAudioData**

Core method for streaming audio.

Receives the total file size from the server and then enters a loop to receive and play audio data chunks.

Switches between the two secondary buffers to maintain continuous playback.

Checks for pause status and waits for the current buffer to finish playing before loading new data.

**Method: LoadBuffer**

Loads audio data into a given DirectSound buffer.

Locks the buffer, copies the data, and then unlocks the buffer.

**Method: PlayBuffer**

Plays the audio data loaded into a given DirectSound buffer.

Checks for failure in starting playback.

**Method: WaitForBuffer**

Waits for a given DirectSound buffer to finish playing.

Continuously checks the buffer status and uses a brief sleep to prevent busy waiting.

**Method: ReceiveAudioChunk**

Receives a chunk of audio data from the server.

Checks for network errors during data reception.

**MusicClient.cpp:**

**This C++ code is part of a music streaming client application, which is designed for connecting to a music server, managing audio streaming, and controlling playback. The key components include network communication, audio management, and multithreading.**

**Constructor**

Initializes Winsock for network communication.

Creates an instance of DirectSoundManager, which is responsible for managing DirectSound audio functionalities.

Sets the initial volume and initializes the clientSocket with INVALID\_SOCKET.

**Destructor**

Ensures the streaming thread is properly joined before the object is destroyed.

Closes the network socket and performs Winsock cleanup.

**ConnectAndPrepareStreamer**

Establishes a socket connection to the specified music server.

Handles errors for socket creation and connection attempts.

Initializes audioStreamer, passing the client socket and two secondary buffers from DirectSoundManager.

Sets the initial volume for the audio streamer.

**ReceiveFileList**

Receives a list of file names from the server.

Uses a loop to continuously receive data until an "end-of-list" marker is encountered.

Converts the received string data into wide strings (std::wstring) and adds them to a file list.

**PauseStreaming**

Pauses or resumes audio streaming using the audioStreamer object.

**SetVolume**

Sets the audio streaming volume.

**RequestAudioTrack**

Sends a request to the server to play a specific audio track.

Converts a wide string (std::wstring) to a narrow string (std::string) for the request.

**StartStreaming**

Starts audio streaming in a separate thread.

Ensures any previously running streaming thread is properly joined before starting a new one.

**AudioClientDLL.cpp**

**This C++ code represents the implementation of a Dynamic Link Library (DLL) for an audio client application. The DLL exposes several functions for managing a music client, which handles tasks such as connecting to a music server, managing audio streams, and controlling playback.**

**Global Instance**

std::unique\_ptr<MusicClient> g\_client;: A global unique pointer to a MusicClient instance, ensuring only one instance exists throughout the DLL's lifecycle.

**DLL Exported Functions**

Using \_\_declspec(dllexport), these functions are made available for external use, typically by other applications or modules.

**InitializeClient**

Initializes the MusicClient with a given window handle (HWND).

Catches and ignores exceptions. This might be improved by logging or handling exceptions more effectively.

**ConnectToServer**

Connects to a music server using the provided IP address and port.

Again, exceptions are caught but not handled explicitly.

**GetFileList**

Retrieves a list of available audio tracks from the server.

Concatenates filenames into a single std::wstring, separated by semicolons.

Provides the number of files via an output parameter (int\* count).

Returns a C-style string (const wchar\_t\*) which is typical for DLL functions to ensure compatibility.

The error handling strategy is basic, simply returning "Error" in case of an exception.

**PauseStreaming**

Pauses or resumes audio streaming based on the boolean flag pause.

Exception handling is present but does not provide detailed feedback.

**SetVolume**

Sets the streaming audio's volume. Accepts a float value between 0.0 and 1.0.

Exception handling exists but lacks detailed error information.

**RequestAudioTrack**

Sends a request to play a specific audio track, identified by trackName.

Includes basic exception handling.

**StartStreaming**

Starts streaming audio data.

Utilizes exception handling.