MUSE: A Remote Music Library

Ryan Houck and David Fletcher

Introduction

MUSE (Musical Universal Serving Environment) is a remote music library that seeks to consolidate large libraries into one central location.

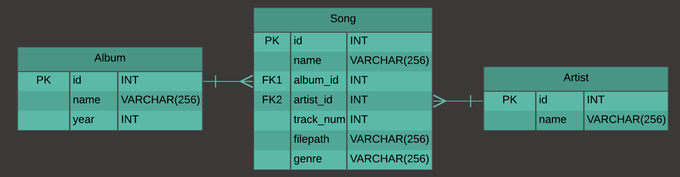
The application consists of two parts: the server and the client. The client asks the server for information about its library or requests a specific song. The server can handle these requests, query its database, and reply with responses of its own.

Architecture

At its core, MUSE is really two behind-the-scenes APIs accessed through C function calls. This abstraction allows other developers to write their own user interfaces, though we provide an implementation for both the client and server side.

Thus, the server and its clients could be run on any machine that has the standard C libraries (plus **TagLib** and **SQLite**) installed on them. A developer only needs to write a user interface suitable for their own application.

Behind the Scenes

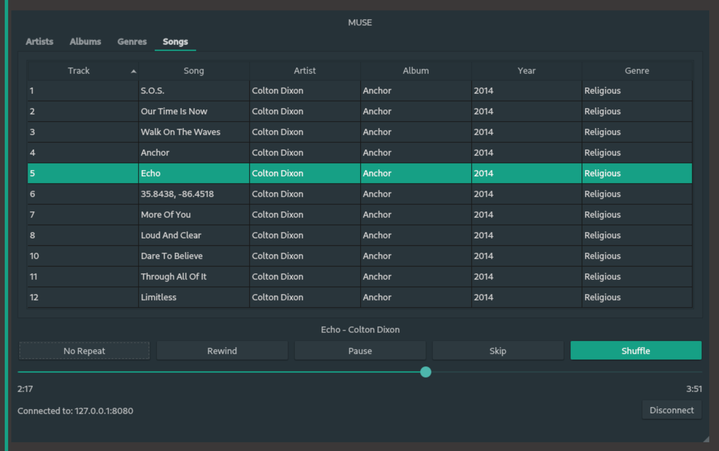


Very early in development, we realized that MUSE had to do some bookkeeping in order to track a sizable audio library efficiently. Due to this, we decided that we should scan the file system for audio files, parse them for relevant information, and populate an **SQLite** database with that information.

In order for the client and server to properly communicate, we realized that we should make a communication protocol (implemented over TCP) to suit our needs. We made the first byte of each request indicate the attribute to order by, how results should be sorted, and the request type (a subset of which is shown below). Additionally, there are other parameters that may be specified if needed. Based upon the received request type, the server will query the database for certain information and send back the requested resources.

Request Code	Description
TERMCON	Terminate the connection
REQSNG	Request a song file
QWRYSNG	Query all songs
QWRYSNGINFO	Query individual song info
QWRYSNGBRST	Query all songs in result range
QWRYALBM	Query all albums
QWRYABLMSNG	Query all songs in album
QWRYALBMBRST	Query all albums in result range
...	...

User Interfaces



The server's user interface had to solve a unique problem. We knew it must be easy to use, accessible from almost anywhere, and lightweight. We chose **ncurses** to solve this problem, as it provides menuing functionality that can also be accessed via an ssh connection. Thus, server owners can access MUSE's core API through our standard interface from wherever they are, even if the server is headless.

The most difficult aspect of using **ncurses** was with more complex menu operations, like a scroll field. We utilized mathematics and careful planning to provide the ability for a user to scroll through a list of items and select one of them.

The client's user interface also had to solve a unique problem: displaying large amounts of tabular data to the user in an understandable way. Additionally, since our client's primary purpose is to play music, we had to find a way to handle many aspects of media playback: pause, skip, rewind, shuffle, etc.

The windowing library we chose, **Qt**, was robust enough to allow us to create the UI relatively quickly while also providing helpful libraries for its event-driven model. This model perfectly suited our application. Combined with **FMOD** and a carefully constructed finite state machine, our client UI became capable of media playback in response to any user input events that may occur at any time.