MediaSphere

OPEN SOURCE COMMUNITY

Official Documentation: MediaSphere Music Player

Version 1.0 | **Last Updated:** July 29, 2025

Abstract

The MediaSphere Music Player is a desktop application designed for a seamless and elegant audio playback experience. As a key component of the open-source MediaSphere Suite, it is engineered with Electron.js to deliver a cross-platform solution for managing and listening to local music libraries. The application leverages the `music-metadata` library to extract rich metadata, including cover art, to provide a visually engaging interface. This document provides a comprehensive overview of the project's architecture, features, installation procedures, and component specifications.

1.0 INTRODUCTION

The MediaSphere Music Player offers users a focused and feature-rich platform for enjoying their local music collections. As part of the broader MediaSphere Suite, it adheres to the same principles of open-source development, performance, and user-centric design. The project is actively developed by a community of contributors, including members from GITAM (Deemed to be University).

The primary objective of the MediaSphere Suite is to create a single, unified application for all media formats. The Music Player represents the project's solution for audio playback.

2.0 KEY FEATURES

- **Folder-Based Library:** Imports entire folders of music files to automatically generate a playable queue.
- **Rich Metadata Display:** Fetches and displays ID3 tag information, including song title, artist, album, and embedded cover art.
- **Standard Playback Controls:** Includes essential controls for play, pause, next track, and previous track.
- **Dynamic Playlist UI:** Displays the full music queue, highlighting the currently playing track.
- **Modern, Clean Interface:** A minimalist and dark-themed UI designed for a pleasant and intuitive listening experience.

3.0 INSTALLATION AND EXECUTION

3.1 Prerequisites

A working installation of Node.js is required to run the application.

3.2 Procedure

Execute the following commands in a terminal or command prompt:

Clone the source repository:
git clone https://github.com/AtheeqAhmedMJ/MediaSphereMusicPlayer.git

1.

Navigate to the project directory:
cd MediaSphereMusicPlayer

2.

Install dependencies:
npm install

3.

Execute the application:
npm start

4.

4.0 SYSTEM ARCHITECTURE

The application is built using the **Electron.js** framework, which enables the creation of desktop applications with web technologies. The architecture is bifurcated into two primary processes to ensure security and performance:

1. **Main Process (`main.js`):** The application's backend, running in a Node.js environment. It manages application windows and handles native operating system interactions like reading the file system and parsing music metadata.
2. **Renderer Process (`renderer.js`):** The application's frontend, responsible for rendering the user interface and handling all user interactions within a sandboxed Chromium window.

These processes communicate securely through a **Preload Script (`preload.js`)**, which selectively exposes backend functions to the frontend.

4.1 Data Flow Example: Loading a Music Folder

To understand how the components work together, consider the step-by-step process when a user loads a folder of music:

1. **User Action:** The user clicks the "Open Folder" button in the Renderer Process (the user interface).
2. **Secure API Call:** The UI's JavaScript (`renderer.js`) calls `window.electronAPI.openFolder()`. This function was securely exposed by the Preload Script.
3. **IPC Message:** The Preload Script sends a secure message (`'dialog:openFolder'`) over the IPC channel to the Main Process.
4. **Native Action:** The Main Process (`main.js`) receives the message and opens the operating system's native folder selection dialog.
5. **File System Read:** After the user selects a folder, the Main Process reads the list of all files within that folder.
6. **Metadata Parsing:** It iterates through the files, filtering for audio formats (e.g., `.mp3`). For each audio file, it uses the `music-metadata` library to parse its metadata tags (artist, title, album, cover art).
7. **IPC Response:** The Main Process compiles an array of song objects, each containing the file path and its extracted metadata, and sends this array back to the Renderer Process.
8. **UI Update:** The Renderer Process receives the array of songs and dynamically populates the playlist on the screen, making the tracks available for playback.

5.0 COMPONENT SPECIFICATION

This section details the function and design of each core file in the project.

5.1 Project Manifest (`package.json`)

This file serves as the project's configuration manifest, defining its metadata and dependencies.

- **`"main": "main.js"`**: Specifies the entry point for the Electron application.
- **`"scripts": { "start": "electron ." }`**: Defines the `npm start` command for easy execution.
- **`"dependencies": { "music-metadata": "..." }`**: Declares **music-metadata** as a critical dependency for reading tags from audio files.

5.2 Main Process (`main.js`)

This script controls the application's lifecycle and backend operations, including the heavy lifting of data extraction.

- **IPC Handler: `ipcMain.handle('dialog:openFolder', ...)`**
  - **Purpose:** To provide a secure way for the UI to access the local file system and to offload the time-consuming task of metadata parsing from the UI thread.

    - **Action:** When invoked, this handler opens a folder picker. It then reads the directory's contents, filters for audio files, and iterates through them. For each file, it calls `musicMetadata.parseFile()` to extract the ID3 tags. It also converts the raw image buffer for the cover art into a Base64 data URL that can be easily used in an `<img>` tag in the UI.
    - **Result:** A complete, structured list of song data is prepared in the backend and sent to the UI in a single batch. This prevents the user interface from freezing while the files are being processed.

5.3 Preload Script (`preload.js`)

This script acts as a secure bridge between the frontend and backend.

- **API: `contextBridge.exposeInMainWorld('electronAPI', ...)`**
    - **Purpose:** To securely expose specific backend functions to the renderer process.
    - **Action:** The `contextBridge` module attaches a custom `electronAPI` object to the UI's global `window` object. This object contains the `openFolder()` function, which internally invokes the `dialog:openFolder` IPC handler in the main process.
    - **Result:** The UI can call `window.electronAPI.openFolder()` to trigger the folder selection and metadata parsing process, maintaining a strong security boundary.

5.4 Renderer Process (`renderer.js`)

This script governs the application's user interface, managing audio playback and all user interactions.

- **Playlist Generation**
    - **Purpose:** To display the list of imported songs to the user.
    - **Action:** After the `openFolder` call returns the array of song data, the script iterates through this array. For each song, it dynamically creates a new `<div>` element containing the title and artist, and appends it to the playlist container in the HTML.
    - **Result:** The user sees a complete, interactive list of all the tracks that were successfully loaded from the selected folder.
- **Audio Playback Logic**
    - **Purpose:** To control the playback of the selected audio file using standard browser capabilities.
    - **Action:** The script uses a single HTML `<audio>` element. When a user clicks a song in the playlist, the `src` of this audio element is set to the song's file path. The playback controls (play, pause, etc.) call the corresponding methods on the audio element, such as `audio.play()` and `audio.pause()`.

    - **Result:** The application can play local audio files seamlessly. Using a single audio element is efficient and simplifies state management for playback.
- **UI Synchronization**
    - **Purpose:** To keep the user interface (now playing info, cover art) in sync with the currently playing track.
    - **Action:** When a new song begins to play, its metadata (title, artist, and the Base64 cover art string) is used to update the `innerHTML` and `src` of the corresponding "Now Playing" elements in the UI.
    - **Result:** The user always has a clear visual indication of which song is currently playing, creating a polished and professional user experience.

6.0 LICENSE