Music Player

Generated by Doxygen 1.13.2

Chapter 1

The documentation of Music Player

Welcome to the full technical documentation of music-player - a simple and lightweight UNIX terminal mp3 player app.

Created for the course **NPRG041: Programming in C++** at Charles University.

1.1 Features

- **File explorer**
  - Navigate directories in *midnight commander* style
- **MP3 file playback**
  - Volume control
  - Pause/Resume
- **Track queue**
  - Automatic fetching from queue upon track completion
  - Skip/Add controls

1.1.1 Controls

- `q` — Quit
- `a` — Add track to queue
- `n` — Skip to next queued track
- `Space` — Pause/Resume
- `u/d` — Volume up/down
- `Enter` — Play selected track

1.2 Architecture

The program is divided into multiple components:

- `Decoder` — Loads and processes MP3 files
- `Player` — Manages playback in a background thread
- `UiController` — Handles user input and control flow
- `UiRenderer` — Encapsulates rendering logic using ncurses

Visit architecture.md for more details.

1.3 Dependencies

- `C++ 20`
- `RtAudio` — for real-time audio output
- `minimp3` — for MP3 decoding
- `ncurses` — for terminal UI

Chapter 2

Architecture overview

2.1 MP3 Playback Handling

2.1.1 Decoder

The decoder class implements the minimp3 library.
Its purpose is to decode an MP3 file and return a **Track** object.

2.1.2 Player

The player class handles **Track** objects.
It first needs to load a **Track** (Player::load_track(track))
Its main method - Player::play_track() is intended to be used on a separate thread.

- The method can be then controlled from the outside using the public interface e.g. Player::pause_track()

2.1.3 Track

A set of classes that hold track info

- General track information - metadata
- Data crucial for playback - sample rate, PCM data, channel count

2.2 UI

The UI of the app is created with the support of ncurses

2.2.1 UiController

Runs the main render loop for the entire user interface
Takes care of user input
Evaluates current states and delegates rendering jobs to the renderer

2.2.2 UiRenderer

Helper class for the UiController
Receives data from the controller and updates specific windows of the app

Chapter 3

Project Specification

3.1 UNIX Terminal Music Player

3.2 Goals:

Implement a music player capable of playing MP3 files operating in the UNIX command line interface.
Allow the user to have control over the music - pause, play, skip, add to queue, volume control.
Make the app compatible with Linux and macOS systems.

3.3 Functionalities:

1. MP3 Playback
    - Load and decode an MP3 file
    - Send PCM data to audio hardware API
2. Text-based user interface (TUI)
    - Intuitive and easy-to-use controls for file playing
    - Design inspired by GNU Midnight Commander
3. Display current track information
    - Metadata from the loaded file
4. Player controls
    - Volume control
    - Pause and play
    - Skip
5. Track queue
    - Skip feature grabs tracks from the queue

3.4 Libraries:

1. MP3 File Handling
    - MP3 files are encoded in a very complex way. This means I need to use an external library to decode them.
    - **minimp3**:
        - minimp3 is a very minimalistic single-header library written in C

2. Audio hardware API
    - For the communication with hardware I will be using **rtaudio**
    - It is a set of C++ classes that will allow me to open a stream for the PCM data on both Linux and macOS
3. User interface
    - To achieve the look and feel of Midnight Commander I chose to use **ncurses**

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 7

Class Documentation

7.1 AudioData Struct Reference

Contains PCM audio samples and playback-related data.
`#include <decoder.hpp>`

Public Attributes

- int16_t * pcmData
  Raw track PCM data.
- size_t total_samples
- size_t current_sample
  Current position during playback.
- int channels
- float volume = DEFAULT_VOLUME

7.1.1 Detailed Description

Contains PCM audio samples and playback-related data.
Definition at line 29 of file decoder.hpp.

7.1.2 Member Data Documentation

7.1.2.1 channels

`int AudioData::channels`
Definition at line 33 of file decoder.hpp.

7.1.2.2 current_sample

`size_t AudioData::current_sample`
Current position during playback.
Definition at line 32 of file decoder.hpp.

7.1.2.3 pcmData

`int16_t* AudioData::pcmData`
Raw track PCM data.
Definition at line 30 of file decoder.hpp.

7.1.2.4 total_samples

`size_t AudioData::total_samples`
Definition at line 31 of file decoder.hpp.

7.1.2.5 volume

```
float AudioData::volume = DEFAULT_VOLUME
```

Definition at line 34 of file decoder.hpp.

The documentation for this struct was generated from the following file:

- src/decoder.hpp

7.2 Decoder Class Reference

Transforms valid file into PCM data and all necessary parameters for audio playback.

```
#include <decoder.hpp>
```

Static Public Member Functions

- static track_ptr_t decode_mp3 (const name_t &track_name)

  Attempts to decode a file as an MP3.

Static Private Member Functions

- static MetaData parse_id3v1 (const name_t &file_name)

  Parses ID3v1 metadata from the file.

7.2.1 Detailed Description

Transforms valid file into PCM data and all necessary parameters for audio playback.

Definition at line 65 of file decoder.hpp.

7.2.2 Member Function Documentation

7.2.2.1 decode_mp3()

```
track_ptr_t Decoder::decode_mp3 (
            const name_t & track_name)  [static]
```

Attempts to decode a file as an MP3.

Parameters

track_name	Path to target file

Returns

Pointer to a Track object, if decoding failed target is an ErrorTrack

Definition at line 10 of file decoder.cpp.

```
00011 {
00012     FILE *f = fopen(track_name.c_str(), "rb");
00013     if (!f)
00014     {
00015         return std::make_shared<ErrorTrack>("Failed to open file");
00016     }
00017     fseek(f, 0, SEEK_END);
00018     size_t filesize = ftell(f);
00019     fseek(f, 0, SEEK_SET);
00020
00021     uint8_t* mp3_data = (uint8_t*)malloc(filesize);
00022     fread(mp3_data, filesize, 1, f);
00023     fclose(f);
00024
00025     mp3dec_ex_t mp3dec;
00026     if (mp3dec_ex_open_buf(&mp3dec, mp3_data, filesize, MP3D_SEEK_TO_SAMPLE))
00027     {
00028         free(mp3_data);
00029         return std::make_shared<ErrorTrack>("Failed to open mp3 file");
00030     }
```

```
00031
00032     size_t pcmBufferSize = mp3dec.samples * sizeof(int16_t);
00033     int16_t* pcmData = (int16_t*)malloc(pcmBufferSize);
00034     if(!pcmData)
00035     {
00036         mp3dec_ex_close(&mp3dec);
00037         free(mp3_data);
00038         return std::make_shared<ErrorTrack>("Failed to allocate memory");
00039     }
00040
00041     size_t samplesRead = mp3dec_ex_read(&mp3dec, pcmData, mp3dec.samples);
00042     if (samplesRead == 0)
00043     {
00044         free(pcmData);
00045         mp3dec_ex_close(&mp3dec);
00046         free(mp3_data);
00047         return std::make_shared<ErrorTrack>("No samples decoded");
00048     }
00049
00050     AudioData audioData = {pcmData, samplesRead, 0, mp3dec.info.channels};
00051
00052     // GET METADATA
00053     MetaData metaData = parse_id3v1(track_name);
00054     metaData.duration = audioData.total_samples / (mp3dec.info.hz * audioData.channels);
00055
00056     return std::make_shared<MP3Track>(metaData, audioData, mp3dec.info.hz);
00057 };
```

7.2.2.2 parse_id3v1()

```
MetaData Decoder::parse_id3v1 (
            const name_t & file_name)  [static], [private]
```

Parses ID3v1 metadata from the file.

Parameters

file_name	Path to the MP3 file.

Returns

MetaData structure containing track info.

Definition at line 59 of file decoder.cpp.

```
00059                                                   {
00060     std::filesystem::path fallback_name(filename);
00061     FILE* f = fopen(filename.c_str(), "rb");
00062     if (!f) {
00063         perror("Failed to open file");
00064         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00065     }
00066
00067     // Seek to the last 128 bytes - ID3v1 location
00068     if (fseek(f, -128, SEEK_END) != 0) {
00069         fclose(f);
00070         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00071     }
00072
00073     char tag[128] = {0};
00074     if (fread(tag, 1, 128, f) != 128) {
00075         fclose(f);
00076         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00077     }
00078     fclose(f);
00079
00080     // Validate Tag
00081     if (strncmp(tag, "TAG", 3) != 0) {
00082         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00083     }
00084
00085     MetaData meta;
00086     meta.track_name = std::string(tag + 3, 30);
00087     meta.artist = std::string(tag + 33, 30);
00088     meta.album = std::string(tag + 63, 30);
00089
00090     meta.track_name.erase(meta.track_name.find_last_not_of('\0') + 1);
00091     meta.artist.erase(meta.artist.find_last_not_of('\0') + 1);
00092     meta.album.erase(meta.album.find_last_not_of('\0') + 1);
00093
00094     return meta;
```

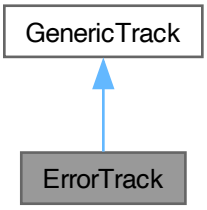
```
00095 }
```

The documentation for this class was generated from the following files:

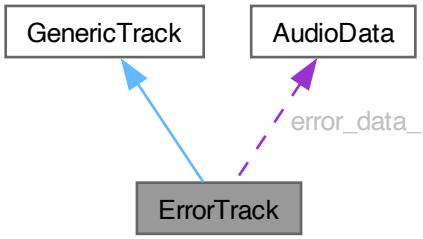
- src/decoder.hpp
- src/decoder.cpp

7.3 ErrorTrack Class Reference

Inheritance diagram for ErrorTrack:



Collaboration diagram for ErrorTrack:



Public Member Functions

- ErrorTrack (const name_t &message)
- TrackInfo getTrackInfo () const override

  Returns a copy of the track's metadata and audio data.
- AudioData & getAudioDataRef () override

  Returns a reference to actual AudioData for tracking playback progress.
- void setCurrentSample (const size_t &position) override

Private Attributes

- AudioData error_data_ = AudioData()
- name_t error_message_

7.3.1 Detailed Description

Definition at line 113 of file decoder.hpp.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 ErrorTrack()

```
ErrorTrack::ErrorTrack (
            const name_t & message)  [inline], [explicit]
```

Definition at line 115 of file decoder.hpp.

```
00116         : error_message_(message) {}
```

7.3.3 Member Function Documentation

7.3.3.1 getAudioDataRef()

```
AudioData & ErrorTrack::getAudioDataRef ()  [inline], [override], [virtual]
```

Returns a reference to actual AudioData for tracking playback progress.
Implements GenericTrack.
Definition at line 118 of file decoder.hpp.

```
00118 { return error_data_; }
```

7.3.3.2 getTrackInfo()

```
TrackInfo ErrorTrack::getTrackInfo () const  [inline], [override], [virtual]
```

Returns a copy of the track's metadata and audio data.
Implements GenericTrack.
Definition at line 117 of file decoder.hpp.

```
00117 { return TrackInfo {error_message_, "", "",0, AudioData()};}
```

7.3.3.3 setCurrentSample()

```
void ErrorTrack::setCurrentSample (
            const size_t & position)  [inline], [override], [virtual]
```

Implements GenericTrack.
Definition at line 119 of file decoder.hpp.

```
00119 {}
```

7.3.4 Member Data Documentation

7.3.4.1 error_data_

```
AudioData ErrorTrack::error_data_ = AudioData()  [private]
```

Definition at line 121 of file decoder.hpp.

7.3.4.2 error_message_

```
name_t ErrorTrack::error_message_  [private]
```

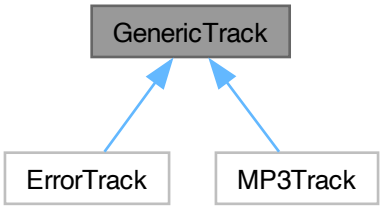
Definition at line 122 of file decoder.hpp.
The documentation for this class was generated from the following file:

- src/decoder.hpp

7.4 GenericTrack Class Reference

Abstract base class for all track types.
`#include <decoder.hpp>`

Inheritance diagram for GenericTrack:



Public Member Functions

- virtual TrackInfo getTrackInfo () const =0

  Returns a copy of the track's metadata and audio data.
- virtual AudioData & getAudioDataRef ()=0

  Returns a reference to actual AudioData for tracking playback progress.
- virtual void **setCurrentSample** (const size_t &position)=0

7.4.1 Detailed Description

Abstract base class for all track types.
Definition at line 86 of file decoder.hpp.

7.4.2 Member Function Documentation

7.4.2.1 getAudioDataRef()

```
virtual AudioData & GenericTrack::getAudioDataRef ()  [pure virtual]
```

Returns a reference to actual AudioData for tracking playback progress.
Implemented in ErrorTrack, and MP3Track.

7.4.2.2 getTrackInfo()

```
virtual TrackInfo GenericTrack::getTrackInfo () const  [pure virtual]
```

Returns a copy of the track's metadata and audio data.
Implemented in ErrorTrack, and MP3Track.
The documentation for this class was generated from the following file:

- src/decoder.hpp

7.5 MetaData Struct Reference

Holds data related to track context.
`#include <decoder.hpp>`

Public Attributes

- name_t track_name
- name_t artist
- name_t album
- uint32_t duration

7.5.1 Detailed Description

Holds data related to track context.
Definition at line 41 of file decoder.hpp.

7.5.2 Member Data Documentation

7.5.2.1 album

`name_t MetaData::album`
Definition at line 44 of file decoder.hpp.

7.5.2.2 artist

`name_t MetaData::artist`
Definition at line 43 of file decoder.hpp.

7.5.2.3 duration

`uint32_t MetaData::duration`
Definition at line 45 of file decoder.hpp.

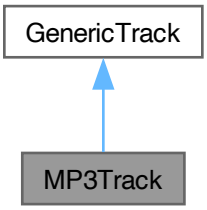
7.5.2.4 track_name

`name_t MetaData::track_name`
Definition at line 42 of file decoder.hpp.
The documentation for this struct was generated from the following file:

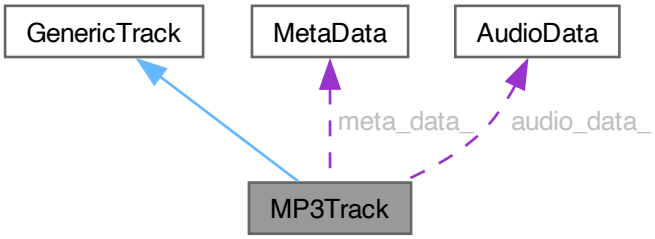
- src/decoder.hpp

7.6 MP3Track Class Reference

Inheritance diagram for MP3Track:



Collaboration diagram for MP3Track:



Public Member Functions

- MP3Track (const MetaData &meta_data, const AudioData &data, const unsigned int sample_rate)
- TrackInfo getTrackInfo () const override

  Returns a copy of the track's metadata and audio data.
- AudioData & getAudioDataRef () override

  Returns a reference to actual AudioData for tracking playback progress.
- void setCurrentSample (const size_t &position) override

Private Attributes

- MetaData meta_data_
- AudioData audio_data_
- unsigned int sample_rate_

7.6.1 Detailed Description

Definition at line 100 of file decoder.hpp.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MP3Track()

```
MP3Track::MP3Track (
            const MetaData & meta_data,
            const AudioData & data,
            const unsigned int sample_rate)  [inline], [explicit]
```

Definition at line 102 of file decoder.hpp.

```
00103         : meta_data_(meta_data), audio_data_(data), sample_rate_(sample_rate) {}
```

7.6.3 Member Function Documentation

7.6.3.1 getAudioDataRef()

```
AudioData & MP3Track::getAudioDataRef ()  [inline], [override], [virtual]
```

Returns a reference to actual AudioData for tracking playback progress.
Implements GenericTrack.
Definition at line 105 of file decoder.hpp.

```
00105 { return audio_data_; }
```

7.6.3.2 getTrackInfo()

```
TrackInfo MP3Track::getTrackInfo () const  [inline], [override], [virtual]
```

Returns a copy of the track's metadata and audio data.
Implements GenericTrack.
Definition at line 104 of file decoder.hpp.

```
00104 { return TrackInfo {meta_data_, audio_data_, sample_rate_};}
```

7.6.3.3 setCurrentSample()

```
void MP3Track::setCurrentSample (
            const size_t & position)  [inline], [override], [virtual]
```

Implements GenericTrack.
Definition at line 106 of file decoder.hpp.

```
00106 { audio_data_.current_sample = position; }
```

7.6.4 Member Data Documentation

7.6.4.1 audio_data_

```
AudioData MP3Track::audio_data_  [private]
```

Definition at line 109 of file decoder.hpp.

7.6.4.2 meta_data_

```
MetaData MP3Track::meta_data_  [private]
```

Definition at line 108 of file decoder.hpp.

7.6.4.3 sample_rate_

```
unsigned int MP3Track::sample_rate_  [private]
```

Definition at line 110 of file decoder.hpp.
The documentation for this class was generated from the following file:

- src/decoder.hpp

7.7 Player Class Reference

Manages playback of data from a Track instance.
`#include <player.hpp>`

Public Member Functions

- Player ()

  Constructs a new Player instance and initializes audio system.
- void load_track (const track_ptr_t &track)

  Redirects local pointer to given Track instance.
- void play_track ()

  Starts track playback.
- void raise_volume ()
- void lower_volume ()
- void pause_track ()
- void resume_track ()
- bool is_paused () const
- void stop_track ()

  Only way to abort the execution of play_track() method from outside of Player class.
- bool is_stopped () const

Static Private Member Functions

- static int audioCallback (void *outputBuffer, void *, unsigned int nFrames, double, RtAudioStreamStatus, void *userData)

Private Attributes

- std::atomic< bool > stop_playback_ {false}
- bool paused_
- RtAudio audio_
- RtAudio::StreamParameters params_
- unsigned int BUFFER_SIZE = 512
- unsigned int SAMPLE_RATE = 44100

  Audio sample rate in Hz.
- float saved_volume_ = DEFAULT_VOLUME

  Saves volume modifier when switching between tracks.
- track_ptr_t track_

  Current Track in playback or ready to be played.

7.7.1 Detailed Description

Manages playback of data from a Track instance.
Reads data from a valid Track instance and feeds the raw PCM into an audio stream Has multiple states:

- Playing - audio stream is open
- Paused - the player is stuck in the play_track() method and awaits resume
- Stopped - playback aborts and Player returns to the initial state

Definition at line 29 of file player.hpp.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Player()

```
Player::Player ()  [explicit]
```

Constructs a new Player instance and initializes audio system.
Definition at line 6 of file player.cpp.

```
00006                : paused_(false)
00007 {
00008     if (audio_.getDeviceCount() == 0) {
00009         throw std::runtime_error("No audio devices found!");
00010     }
00011
00012     params_.deviceId = audio_.getDefaultOutputDevice();
00013     params_.nChannels = 1;
00014     params_.firstChannel = 0;
00015 }
```

7.7.3 Member Function Documentation

7.7.3.1 audioCallback()

```
int Player::audioCallback (
             void * outputBuffer,
             void * ,
             unsigned int nFrames,
             double ,
             RtAudioStreamStatus ,
             void * userData)  [static], [private]
```

Definition at line 119 of file player.cpp.

```
00120                                                    {
```

```
00121     auto *audio_data = static_cast<AudioData *>(userData);
00122     if(!audio_data) return 0;
00123
00124     int16_t *buffer = static_cast<int16_t *>(outputBuffer);
00125     size_t channels = audio_data->channels;
00126     float volume = audio_data->volume;
00127
00128     for (unsigned int i = 0; i < nFrames; i++) {
00129         for (size_t ch = 0; ch < channels; ch++) {
00130             if (audio_data->current_sample < audio_data->total_samples) {
00131                 buffer[i * channels + ch] = static_cast<int16_t>(
00132                     audio_data->pcmData[audio_data->current_sample++] * volume);
00133             } else {
00134                 buffer[i * channels + ch] = 0;
00135             }
00136         }
00137     }
00138     return 0;
00139 }
```

7.7.3.2 is_paused()

```
bool Player::is_paused () const  [nodiscard]
```

Definition at line 28 of file player.cpp.

```
00029 {
00030     return paused_;
00031 }
```

7.7.3.3 is_stopped()

```
bool Player::is_stopped () const
```

Definition at line 22 of file player.cpp.

```
00023 {
00024     return stop_playback_;
00025 }
```

7.7.3.4 load_track()

```
void Player::load_track (
            const track_ptr_t & track)
```

Redirects local pointer to given Track instance.

Parameters

track	A valid! Track instance

Definition at line 46 of file player.cpp.

```
00047 {
00048     track_ = track;
00049     paused_ = false;
00050 }
```

7.7.3.5 lower_volume()

```
void Player::lower_volume ()
```

Definition at line 61 of file player.cpp.

```
00062 {
00063     if (track_)
00064     {
00065         AudioData& audio_data = track_->getAudioDataRef();
00066         audio_data.volume = std::clamp(audio_data.volume - 0.02f, 0.0f, 1.0f);
00067         saved_volume_ = audio_data.volume;
00068     }
00069 }
```

7.7.3.6 pause_track()

```
void Player::pause_track ()
```

Definition at line 34 of file player.cpp.

```
00035 {
```

```
00036     paused_ = true;
00037 }
```

7.7.3.7 play_track()

```
void Player::play_track ()
```

Starts track playback.

Warning

Blocking method, should be called in a new thread

First call load_track() to prime the method, otherwise no playback will proceed

Starts track playback and only completes after stop_track() has been called or Track reached the end Uses the rtaudio interface to open a stream to the primary audio output device Can be controller from other threads by utilizing the rest of the public interface e.g. pause_track()
Definition at line 72 of file player.cpp.

```
00073 {
00074     stop_playback_ = false;
00075
00076     if (track_ == nullptr)
00077     {
00078         throw std::runtime_error("No track found!");
00079     }
00080     if (audio_.isStreamOpen()) {
00081         std::cout « "Closing previous audio stream..." « std::endl;
00082         audio_.closeStream();
00083     }
00084
00085     TrackInfo info = track_->getTrackInfo();
00086     name_t name = info.meta_data.track_name;
00087     SAMPLE_RATE = info.sample_rate;
00088     params_.nChannels = info.data.channels;
00089     AudioData& audio_data = track_->getAudioDataRef();
00090     audio_data.volume = saved_volume_;
00091
00092     try {
00093         audio_.openStream(&params_, nullptr, RTAUDIO_SINT16, SAMPLE_RATE, &BUFFER_SIZE, audioCallback,
      &audio_data);
00094         audio_.startStream();
00095         while (audio_.isStreamRunning() && !stop_playback_) {
00096             if (audio_data.current_sample >= audio_data.total_samples) {
00097                 stop_playback_ = true;
00098             }
00099             if (is_paused())
00100                 audio_.stopStream();
00101
00102             while (is_paused() && !stop_playback_)
00103             {
00104                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
00105             }
00106
00107             if (!audio_.isStreamRunning() && !stop_playback_)
00108             {
00109                 audio_.startStream();
00110             }
00111             std::this_thread::sleep_for(std::chrono::milliseconds(100));
00112         }
00113         audio_.closeStream();
00114     } catch (RtAudioErrorType &e) {
00115         if (audio_.isStreamOpen()) audio_.closeStream();
00116     }
00117 }
```

7.7.3.8 raise_volume()

```
void Player::raise_volume ()
```

Definition at line 52 of file player.cpp.

```
00053 {
00054     if (track_)
00055     {
00056         AudioData& audio_data = track_->getAudioDataRef();
00057         audio_data.volume = std::clamp(audio_data.volume + 0.02f, 0.0f, 1.0f);
00058         saved_volume_ = audio_data.volume;
00059     }
00060 }
```

7.7.3.9 resume_track()

```
void Player::resume_track ()
```

Definition at line 39 of file player.cpp.

```
00040 {
00041     if (paused_) {
00042         paused_ = false;
00043     }
00044 }
```

7.7.3.10 stop_track()

```
void Player::stop_track ()
```

Only way to abort the execution of play_track() method from outside of Player class.
Definition at line 17 of file player.cpp.

```
00018 {
00019     stop_playback_ = true;
00020 }
```

7.7.4 Member Data Documentation

7.7.4.1 audio_

```
RtAudio Player::audio_  [private]
```

Definition at line 70 of file player.hpp.

7.7.4.2 BUFFER_SIZE

```
unsigned int Player::BUFFER_SIZE = 512  [private]
```

Definition at line 72 of file player.hpp.

7.7.4.3 params_

```
RtAudio::StreamParameters Player::params_  [private]
```

Definition at line 71 of file player.hpp.

7.7.4.4 paused_

```
bool Player::paused_  [private]
```

Definition at line 69 of file player.hpp.

7.7.4.5 SAMPLE_RATE

```
unsigned int Player::SAMPLE_RATE = 44100  [private]
```

Audio sample rate in Hz.
Definition at line 73 of file player.hpp.

7.7.4.6 saved_volume_

```
float Player::saved_volume_ = DEFAULT_VOLUME  [private]
```

Saves volume modifier when switching between tracks.
Definition at line 74 of file player.hpp.

7.7.4.7 stop_playback_

```
std::atomic<bool> Player::stop_playback_ {false}  [private]
```

Definition at line 68 of file player.hpp.

```
00068 {false};
```

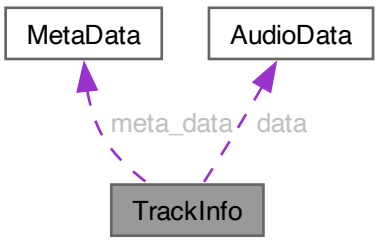
7.7.4.8 track_

`track_ptr_t Player::track_ [private]`
Current Track in playback or ready to be played.
Definition at line 75 of file player.hpp.
The documentation for this class was generated from the following files:

- src/player.hpp
- src/player.cpp

7.8 TrackInfo Struct Reference

Couples together playback and context information for a specific track.
`#include <decoder.hpp>`
Collaboration diagram for TrackInfo:



Public Attributes

- MetaData meta_data
- AudioData data
- uint32_t sample_rate

7.8.1 Detailed Description

Couples together playback and context information for a specific track.
Definition at line 52 of file decoder.hpp.

7.8.2 Member Data Documentation

7.8.2.1 data

`AudioData TrackInfo::data`
Definition at line 54 of file decoder.hpp.

7.8.2.2 meta_data

`MetaData TrackInfo::meta_data`
Definition at line 53 of file decoder.hpp.

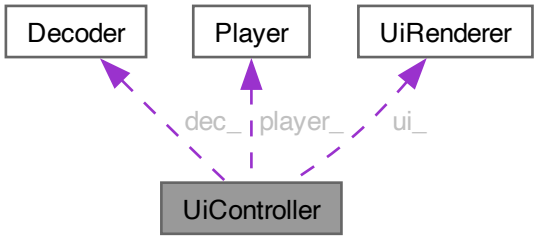
7.8.2.3 sample_rate

`uint32_t TrackInfo::sample_rate`
Definition at line 55 of file decoder.hpp.
The documentation for this struct was generated from the following file:

- src/decoder.hpp

7.9 UiController Class Reference

Takes care of user input handling and interaction between program components.
`#include <ui_controller.hpp>`
Collaboration diagram for UiController:



Public Member Functions

- UiController ()

  Initializes ncurses core settings.
- void beginRenderLoop ()

  Starts main loop of user interface.

Private Member Functions

- void beginTrackPlayback (const track_ptr_t &track)
- void stopTrackPlayback ()
- void updateFileList ()
- void showErrorPopup (const std::string &message) const
- void processTrackSelection ()

  Attempt to decode a file and start playback.
- void addTrackToQueue ()
- void processNextTrackFromQueue ()

Private Attributes

- Decoder dec_

  Decoder instance for creating Track objects.
- Player player_

  Player instance to delegate Track playback.
- UiRenderer ui_

  Helper renderer instance that separates concern for ncurses rendering.
- file_list_t files_

  Files in current directory.
- queue_t track_queue_

  Tracks ready to be played.
- track_ptr_t current_track_

Track currently loaded in the player_ instance and playing.

- std::string path_

  Current working directory.

- int highlight_ = 0

  Current cursor position.

- std::thread playback_thread_

  Thread on which the player_ instance operates.

- bool playing_ = false

7.9.1 Detailed Description

Takes care of user input handling and interaction between program components.
The UiController is responsible for:

- Capturing user keyboard input
- Managing track playback
- Delegating UI rendering tasks to UiRenderer
- Using Decoder and Player to start playback

Definition at line 44 of file ui_controller.hpp.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 UiController()

```
UiController::UiController ()
```

Initializes ncurses core settings.
Definition at line 6 of file ui_controller.cpp.

```
00007 {
00008     initscr();
00009     refresh();
00010     noecho();
00011     cbreak();
00012     keypad(stdscr, true);
00013     start_color();
00014     init_pair(1, COLOR_CYAN, COLOR_BLACK);
00015     curs_set(0);
00016     // Explicit initialization of the renderer after all setup ncurses functions have been called
00017     ui_ = UiRenderer();
00018 }
```

7.9.3 Member Function Documentation

7.9.3.1 addTrackToQueue()

```
void UiController::addTrackToQueue ()  [private]
```

Definition at line 168 of file ui_controller.cpp.

```
00169 {
00170     name_t track_path = files_[highlight_].path().string();
00171     track_ptr_t track = dec_.decode_mp3(track_path);
00172     if (dynamic_cast<ErrorTrack*>(track.get()) != nullptr) //Check type of returned track
00173     {
00174         showErrorPopup("Error opening file!");
00175         return;
00176     }
00177     track_queue_.emplace_back(track);
00178     ui_.renderTrackQueue(track_queue_);
00179 }
```

7.9.3.2 beginRenderLoop()

```
void UiController::beginRenderLoop ()
```

Starts main loop of user interface.

Warning

Alive for the entire lifespan of the program

Definition at line 20 of file ui_controller.cpp.

```
00021 {
00022     path_ = std::filesystem::current_path().string();
00023 
00024     updateFileList();
00025     nodelay(stdscr, TRUE);
00026     ui_.renderFileList(files_, highlight_);
00027     ui_.renderTrackQueue(track_queue_);
00028     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00029 
00030     bool running = true;
00031     while (running)
00032     {
00033         ui_.updateAnimationFrame(current_track_, playing_, player_.is_paused());
00034         napms(50);
00035         if (!playing_ && !track_queue_.empty())
00036         {
00037             processNextTrackFromQueue();
00038         }
00039 
00040         switch (int pressed_key = getch()) {
00041             case KEY_UP:
00042                 if (highlight_ > 0) highlight_--;
00043                 ui_.renderFileList(files_, highlight_);
00044                 break;
00045             case KEY_DOWN:
00046                 if (highlight_ < files_.size() - 1) highlight_++;
00047                 ui_.renderFileList(files_, highlight_);
00048                 break;
00049             case KEY_PLAY_TRACK:
00050                 processTrackSelection();
00051                 break;
00052             case KEY_ADD_QUEUE:
00053                 addTrackToQueue();
00054                 break;
00055             case KEY_PAUSE:
00056                 if (playing_) {
00057                     player_.is_paused() ? player_.resume_track() : player_.pause_track();
00058                 }
00059                 ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00060                 break;
00061             case KEY_NEXT_QUEUE:
00062                 processNextTrackFromQueue();
00063                 break;
00064             case KEY_VOLUME_UP:
00065                 player_.raise_volume();
00066                 break;
00067             case KEY_VOLUME_DOWN:
00068                 player_.lower_volume();
00069                 break;
00070             case KEY_QUIT:
00071                 stopTrackPlayback();
00072                 running = false;
00073                 break;
00074         }
00075     }
00076     endwin();
00077 }
```

7.9.3.3 beginTrackPlayback()

```
void UiController::beginTrackPlayback (
            const track_ptr_t & track)  [private]
```

Definition at line 79 of file ui_controller.cpp.

```
00080 {
00081     if (playback_thread_.joinable())
00082     {
00083         playback_thread_.join();
00084     }
00085     player_.load_track(track);
00086     playing_ = true;
00087     current_track_ = track;
00088     playback_thread_ = std::thread([this]() {
00089         player_.play_track();
00090         playing_ = false;
00091         current_track_ = nullptr;
00092         ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00093     });
00094 }
```

7.9.3.4 processNextTrackFromQueue()

```
void UiController::processNextTrackFromQueue ()  [private]
```

Definition at line 181 of file ui_controller.cpp.

```
00182 {
00183     if (playing_)
00184         stopTrackPlayback();
00185     if (!track_queue_.empty())
00186     {
00187         auto next_track = track_queue_.front();
00188         track_queue_.pop_front();
00189         beginTrackPlayback(next_track);
00190     }
00191     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00192     ui_.renderTrackQueue(track_queue_);
00193 }
```

7.9.3.5 processTrackSelection()

```
void UiController::processTrackSelection ()  [private]
```

Attempt to decode a file and start playback.
Handles logic behind the user pressing the KEY_PLAY_TRACK button That includes:

- Moving in and out of directories inside the file explorer
- Attempting to decode the currently selected file in the explorer

Definition at line 139 of file ui_controller.cpp.

```
00140 {
00141     auto& selected = files_[highlight_];
00142     std::filesystem::path selected_path = selected.path();
00143 
00144     if (selected_path.filename() == "..") {
00145         path_ = std::filesystem::path(path_).parent_path().string();
00146         highlight_ = 0;
00147         updateFileList();
00148     } else if (is_directory(selected_path)) {
00149         path_ = selected_path.string();
00150         highlight_ = 0;
00151         updateFileList();
00152     } else
00153     {
00154         if (playing_)
00155             stopTrackPlayback();
00156         track_ptr_t track = dec_.decode_mp3(selected_path.string());
00157         if (dynamic_cast<ErrorTrack*>(track.get()) != nullptr){
00158             //Check type of returned track
00159             showErrorPopup("Error opening file!");
00160             return;
00161         }
00162         beginTrackPlayback(track);
00163     }
00164     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00165     ui_.renderFileList(files_, highlight_);
00166 }
```

7.9.3.6 showErrorPopup()

```
void UiController::showErrorPopup (
            const std::string & message) const  [private]
```

Definition at line 117 of file ui_controller.cpp.

```
00118 {
00119     size_t height = 3;
00120     size_t width = message.size() + 4;
00121     size_t starty = (LINES - height) / 2;
00122     size_t startx = (COLS - width) / 2;
00123 
00124     WINDOW* popup = newwin(height, width, starty, startx);
00125     box(popup, 0, 0);
00126     mvwprintw(popup, 1, 2, "%s", message.c_str());
00127     wrefresh(popup);
00128 
00129     napms(1500);
00130 
00131     delwin(popup);
00132     clear();
00133     refresh();
00134     ui_.renderFileList(files_, highlight_);
```

```
00135     ui_.renderTrackQueue(track_queue_);
00136     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00137 }
```

7.9.3.7 stopTrackPlayback()

```
void UiController::stopTrackPlayback ()  [private]
```

Definition at line 95 of file ui_controller.cpp.

```
00096 {
00097     if (playing_)
00098     {
00099         player_.stop_track();
00100         if (playback_thread_.joinable())
00101             playback_thread_.join();
00102         playing_ = false;
00103         current_track_ = nullptr;
00104     }
00105 }
```

7.9.3.8 updateFileList()

```
void UiController::updateFileList ()  [private]
```

Definition at line 107 of file ui_controller.cpp.

```
00108 {
00109     files_.clear();
00110     files_.emplace_back(std::filesystem::directory_entry(std::filesystem::path(path_) / ".."));
00111     for (const auto& entry : std::filesystem::directory_iterator(path_))
00112     {
00113         files_.push_back(entry);
00114     }
00115 }
```

7.9.4 Member Data Documentation

7.9.4.1 current_track_

```
track_ptr_t UiController::current_track_  [private]
```

Track currently loaded in the player_ instance and playing.
Definition at line 78 of file ui_controller.hpp.

7.9.4.2 dec_

```
Decoder UiController::dec_  [private]
```

Decoder instance for creating Track objects.
Definition at line 73 of file ui_controller.hpp.

7.9.4.3 files_

```
file_list_t UiController::files_  [private]
```

Files in current directory.
Definition at line 76 of file ui_controller.hpp.

7.9.4.4 highlight_

```
int UiController::highlight_ = 0  [private]
```

Current cursor position.
Definition at line 80 of file ui_controller.hpp.

7.9.4.5 path_

```
std::string UiController::path_  [private]
```

Current working directory.
Definition at line 79 of file ui_controller.hpp.

7.9.4.6 playback_thread_

```
std::thread UiController::playback_thread_  [private]
```
Thread on which the player_ instance operates.
Definition at line 81 of file ui_controller.hpp.

7.9.4.7 player_

```
Player UiController::player_  [private]
```
Player instance to delegate Track playback.
Definition at line 74 of file ui_controller.hpp.

7.9.4.8 playing_

```
bool UiController::playing_ = false  [private]
```
Definition at line 82 of file ui_controller.hpp.

7.9.4.9 track_queue_

```
queue_t UiController::track_queue_  [private]
```
Tracks ready to be played.
Definition at line 77 of file ui_controller.hpp.

7.9.4.10 ui_

```
UiRenderer UiController::ui_  [private]
```
Helper renderer instance that separates concern for ncurses rendering.
Definition at line 75 of file ui_controller.hpp.
The documentation for this class was generated from the following files:

- src/ui_controller.hpp
- src/ui_controller.cpp

7.10 UiRenderer Class Reference

Renders updated UI state based on data received from controller class.
`#include <ui_renderer.hpp>`

Public Member Functions

- UiRenderer ()

  Sets up ncurses window sizes based on the current terminal window size.
- void renderFileList (const file_list_t &files, size_t highlight) const

  Refreshes file_list_win_ with provided file data.
- void renderTrackQueue (const std::deque< track_ptr_t > &queue) const

  Refreshes track_queue_win_.
- void renderStatusBar (const track_ptr_t &current_track, const bool &playing, const bool &paused) const

  Renders status bar in its stationary state.
- void refreshAll () const
- void updateAnimationFrame (const track_ptr_t &current_track, const bool &playing, const bool &paused) const

  Updates animations in the status bar window.

Private Member Functions

- void renderTrackPlayingText (const MetaData &meta_data) const

Private Attributes

- WINDOW * file_list_win_

  Main ncurses windows that form the whole app.
- WINDOW * track_queue_win_

  List of Queued Tracks.
- WINDOW * status_bar_win_

  Bottom status bar which displays state information.

Static Private Attributes

- static constexpr int frame_delay = 5

7.10.1 Detailed Description

Renders updated UI state based on data received from controller class.
Definition at line 27 of file ui_renderer.hpp.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 UiRenderer()

```
UiRenderer::UiRenderer ()
```

Sets up ncurses window sizes based on the current terminal window size.
Definition at line 8 of file ui_renderer.cpp.

```
00009 {
00010     int mid_x = COLS / 2;
00011     file_list_win_ = newwin(LINES - 6, mid_x, 0, 0);
00012     track_queue_win_ = newwin(LINES - 6, COLS - mid_x, 0, mid_x);
00013     status_bar_win_ = newwin(6, COLS, LINES - 6, 0);
00014 }
```

7.10.3 Member Function Documentation

7.10.3.1 refreshAll()

```
void UiRenderer::refreshAll () const
```

Definition at line 121 of file ui_renderer.cpp.

```
00122 {
00123     wrefresh(file_list_win_);
00124     wrefresh(track_queue_win_);
00125     wrefresh(status_bar_win_);
00126 }
```

7.10.3.2 renderFileList()

```
void UiRenderer::renderFileList (
            const file_list_t & files,
            size_t highlight) const
```

Refreshes file_list_win_ with provided file data.

Parameters

files	A list of files in the current directory
highlight	Current cursor position

Definition at line 17 of file ui_renderer.cpp.

```
00018 {
00019     wclear(file_list_win_);
00020     box(file_list_win_, 0, 0);
00021     for (size_t i = 0; i < files.size(); ++i) {
00022         if (i == highlight)
00023             wattron(file_list_win_, A_REVERSE);
```

```
00024         if (std::filesystem::path(files[i]).extension() == ".mp3")
00025             wattron(file_list_win_, COLOR_PAIR(1));
00026
00027         if (is_directory(files[i]))
00028             mvwprintw(file_list_win_, i + 1, 2, "/%s", files[i].path().filename().c_str());
00029         else
00030             mvwprintw(file_list_win_, i + 1, 2, files[i].path().filename().c_str());
00031
00032         if (std::filesystem::path(files[i]).extension() == ".mp3")
00033             wattroff(file_list_win_, COLOR_PAIR(1));
00034         if (i == highlight)
00035             wattroff(file_list_win_, A_REVERSE);
00036     }
00037     wrefresh(file_list_win_);
00038 }
```

7.10.3.3 renderStatusBar()

```
void UiRenderer::renderStatusBar (
            const track_ptr_t & current_track,
            const bool & playing,
            const bool & paused) const
```

Renders status bar in its stationary state.

Parameters

current_track	Track that provides metadata to be displayed
playing	
paused	

Definition at line 50 of file ui_renderer.cpp.

```
00051 {
00052     wclear(status_bar_win_);
00053     box(status_bar_win_, 0, 0);
00054     if (playing) {
00055         mvwprintw(status_bar_win_, 0, 2, "Now Playing: ");
00056         mvwprintw(status_bar_win_, 0, 15, "   ");
00057         if (paused)
00058         {
00059             wattron(status_bar_win_, A_BOLD);
00060             mvwprintw(status_bar_win_, 2, 2, "||");
00061         } else {
00062             wattron(status_bar_win_, A_BOLD);
00063             mvwprintw(status_bar_win_, 2, 2, "|>");
00064         }
00065         if (current_track != nullptr)
00066         {
00067             TrackInfo track_info = current_track->getTrackInfo();
00068             renderTrackPlayingText(track_info.meta_data);
00069         }
00070         wattroff(status_bar_win_, A_BOLD);
00071     }
00072     else
00073         mvwprintw(status_bar_win_, 0, 2, "Select a track and press ENTER to play");
00074     mvwprintw(status_bar_win_, 5, 2 , "[Q] Quit  [SPACE] Pause  [A] Add  [N] Next  [U/D] Volume
      Control");
00075     wrefresh(status_bar_win_);
00076 }
```

7.10.3.4 renderTrackPlayingText()

```
void UiRenderer::renderTrackPlayingText (
            const MetaData & meta_data) const  [private]
```

Definition at line 77 of file ui_renderer.cpp.

```
00078 {
00079     wattron(status_bar_win_, A_BOLD);
00080     mvwprintw(status_bar_win_, 2, 6, "%s", metaData.track_name.c_str());
00081     wattroff(status_bar_win_, A_BOLD);
00082     mvwprintw(status_bar_win_, 2, 6 + metaData.track_name.length(), " by: ");
00083     wattron(status_bar_win_, A_BOLD);
00084     mvwprintw(status_bar_win_, 2, 11 + metaData.track_name.length(), "%s", metaData.artist.c_str());
00085     wattroff(status_bar_win_, A_BOLD);
00086     mvwprintw(status_bar_win_, 2, 12 + metaData.track_name.length() + metaData.artist.length(), " on:
      ");
00087     wattron(status_bar_win_, A_BOLD);
```

```
00088     mvwprintw(status_bar_win_, 2, 17 + metaData.track_name.length() + metaData.artist.length(), "%s",
      metaData.album.c_str());
00089     wattroff(status_bar_win_, A_BOLD);
00090 }
```

7.10.3.5 renderTrackQueue()

```
void UiRenderer::renderTrackQueue (
            const std::deque< track_ptr_t > & queue) const
```

Refreshes track_queue_win_.

Parameters

queue	List of tracks that are queued to be played

Definition at line 39 of file ui_renderer.cpp.

```
00040 {
00041     wclear(track_queue_win_);
00042     box(track_queue_win_, 0, 0);
00043     mvwprintw(track_queue_win_, 0, 2, "Track Queue");
00044     for(size_t i = 0; i < queue.size(); ++i)
00045     {
00046         mvwprintw(track_queue_win_, i + 1, 2, "%s",
      queue[i]->getTrackInfo().meta_data.track_name.c_str());
00047     }
00048     wrefresh(track_queue_win_);
00049 }
```

7.10.3.6 updateAnimationFrame()

```
void UiRenderer::updateAnimationFrame (
            const track_ptr_t & current_track,
            const bool & playing,
            const bool & paused) const
```

Updates animations in the status bar window.

Warning

Needs to be updated frequently to achieve fluid movement

Parameters

current_track	Track to update elapsed time counter
playing	Flag between stationary and updating
paused	Decide whether playing animation should be paused

Definition at line 91 of file ui_renderer.cpp.

```
00092 {
00093     static const char *frames[] = {
00094         ".!!!!.!!!!!.", "!!!!!!.!!!!.", "!!!!!!.!!!..", "!!.!!!!.!!!..", "!!..!!!!!.!.",
      "!....!!!..!!", "..!.!...!!.!", "..!!...!!!..", ".!!!...!!!!."
00095     };
00096     static size_t frame = 0;
00097     static size_t frame_counter = 0;
00098     if (playing)
00099     {
00100         if (!paused)
00101         {
00102             frame_counter++;
00103             if (frame_counter >= frame_delay) {
00104                 frame_counter = 0;
00105                 frame = (frame + 1) % std::size(frames);
00106             }
00107         }
00108         if (current_track != nullptr)
00109         {
00110             TrackInfo track_info = current_track->getTrackInfo();
00111             uint32_t elapsed_time_minutes = track_info.data.current_sample / track_info.sample_rate /
      track_info.data.channels / 60;
```

```
00112             uint32_t elapsed_time_seconds = track_info.data.current_sample / track_info.data.channels
      / track_info.sample_rate % 60;
00113             uint32_t total_minutes = track_info.meta_data.duration / 60;
00114             uint32_t total_seconds = track_info.meta_data.duration % 60;
00115             mvwprintw(status_bar_win_, 3, 6, "%d:%02d / %d:%02d",elapsed_time_minutes,
      elapsed_time_seconds, total_minutes, total_seconds);
00116         }
00117         mvwprintw(status_bar_win_, 0, 15, frames[frame]);
00118         wrefresh(status_bar_win_);
00119     }
00120 }
```

7.10.4 Member Data Documentation

7.10.4.1 file_list_win_

`WINDOW* UiRenderer::file_list_win_ [private]`

Main ncurses windows that form the whole app.
File browser window
Definition at line 68 of file ui_renderer.hpp.

7.10.4.2 frame_delay

`int UiRenderer::frame_delay = 5 [static], [constexpr], [private]`

Definition at line 66 of file ui_renderer.hpp.

7.10.4.3 status_bar_win_

`WINDOW* UiRenderer::status_bar_win_ [private]`

Bottom status bar which displays state information.
Definition at line 70 of file ui_renderer.hpp.

7.10.4.4 track_queue_win_

`WINDOW* UiRenderer::track_queue_win_ [private]`

List of Queued Tracks.
Definition at line 69 of file ui_renderer.hpp.
The documentation for this class was generated from the following files:

- src/ui_renderer.hpp
- src/ui_renderer.cpp

Chapter 8

File Documentation

8.1 decoder.cpp

```
00001 //
00002 // Created by Darek Rudiš on 10.02.2025.
00003 //
00004 #include "decoder.hpp"
00005
00006 #include <__filesystem/path.h>
00007
00008 using name_t = std::string;
00009
00010 track_ptr_t Decoder::decode_mp3(const name_t& track_name)
00011 {
00012     FILE *f = fopen(track_name.c_str(), "rb");
00013     if (!f)
00014     {
00015         return std::make_shared<ErrorTrack>("Failed to open file");
00016     }
00017     fseek(f, 0, SEEK_END);
00018     size_t filesize = ftell(f);
00019     fseek(f, 0, SEEK_SET);
00020
00021     uint8_t* mp3_data = (uint8_t*)malloc(filesize);
00022     fread(mp3_data, filesize, 1, f);
00023     fclose(f);
00024
00025     mp3dec_ex_t mp3dec;
00026     if (mp3dec_ex_open_buf(&mp3dec, mp3_data, filesize, MP3D_SEEK_TO_SAMPLE))
00027     {
00028         free(mp3_data);
00029         return std::make_shared<ErrorTrack>("Failed to open mp3 file");
00030     }
00031
00032     size_t pcmBufferSize = mp3dec.samples * sizeof(int16_t);
00033     int16_t* pcmData = (int16_t*)malloc(pcmBufferSize);
00034     if(!pcmData)
00035     {
00036         mp3dec_ex_close(&mp3dec);
00037         free(mp3_data);
00038         return std::make_shared<ErrorTrack>("Failed to allocate memory");
00039     }
00040
00041     size_t samplesRead = mp3dec_ex_read(&mp3dec, pcmData, mp3dec.samples);
00042     if (samplesRead == 0)
00043     {
00044         free(pcmData);
00045         mp3dec_ex_close(&mp3dec);
00046         free(mp3_data);
00047         return std::make_shared<ErrorTrack>("No samples decoded");
00048     }
00049
00050     AudioData audioData = {pcmData, samplesRead, 0, mp3dec.info.channels};
00051
00052     // GET METADATA
00053     MetaData metaData = parse_id3v1(track_name);
00054     metaData.duration = audioData.total_samples / (mp3dec.info.hz * audioData.channels);
00055
00056     return std::make_shared<MP3Track>(metaData, audioData, mp3dec.info.hz);
00057 };
00058
00059 MetaData Decoder::parse_id3v1(const std::string& filename) {
00060     std::filesystem::path fallback_name(filename);
00061     FILE* f = fopen(filename.c_str(), "rb");
```

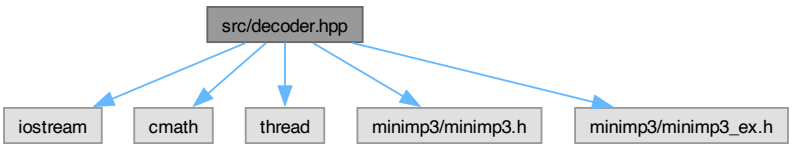
```
00062     if (!f) {
00063         perror("Failed to open file");
00064         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00065     }
00066
00067     // Seek to the last 128 bytes - ID3v1 location
00068     if (fseek(f, -128, SEEK_END) != 0) {
00069         fclose(f);
00070         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00071     }
00072
00073     char tag[128] = {0};
00074     if (fread(tag, 1, 128, f) != 128) {
00075         fclose(f);
00076         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00077     }
00078     fclose(f);
00079
00080     // Validate Tag
00081     if (strncmp(tag, "TAG", 3) != 0) {
00082         return {fallback_name.stem(), "Unknown Artist", "Unknown", 0};
00083     }
00084
00085     MetaData meta;
00086     meta.track_name = std::string(tag + 3, 30);
00087     meta.artist = std::string(tag + 33, 30);
00088     meta.album = std::string(tag + 63, 30);
00089
00090     meta.track_name.erase(meta.track_name.find_last_not_of('\0') + 1);
00091     meta.artist.erase(meta.artist.find_last_not_of('\0') + 1);
00092     meta.album.erase(meta.album.find_last_not_of('\0') + 1);
00093
00094     return meta;
00095 }
```

8.2 src/decoder.hpp File Reference

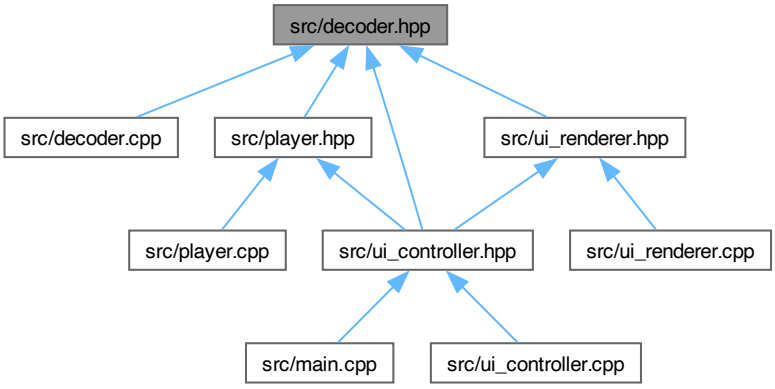
Contains classes and structures for decoding MP3 files into PCM audio data.

```
#include <iostream>
#include <cmath>
#include <thread>
#include "minimp3/minimp3.h"
#include "minimp3/minimp3_ex.h"
```

Include dependency graph for decoder.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct AudioData

  Contains PCM audio samples and playback-related data.
- struct MetaData

  Holds data related to track context.
- struct TrackInfo

  Couples together playback and context information for a specific track.
- class Decoder

  Transforms valid file into PCM data and all necessary parameters for audio playback.
- class GenericTrack

  Abstract base class for all track types.
- class MP3Track
- class ErrorTrack

Typedefs

- using track_ptr_t = std::shared_ptr<GenericTrack>

Variables

- constexpr float DEFAULT_VOLUME = 0.5f

8.2.1 Detailed Description

Contains classes and structures for decoding MP3 files into PCM audio data.

Author

Darek Rudiš

Date

2025-02-10

Definition in file decoder.hpp.

8.2.2 Typedef Documentation

8.2.2.1 track_ptr_t

```
using track_ptr_t = std::shared_ptr<GenericTrack>
```
Definition at line 59 of file decoder.hpp.

8.2.3 Variable Documentation

8.2.3.1 DEFAULT_VOLUME

```
float DEFAULT_VOLUME = 0.5f  [constexpr]
```
Definition at line 22 of file decoder.hpp.

8.3 decoder.hpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Darek Rudiš on 10.02.2025.
00003 //
00010
00011 #include <iostream>
00012
00013 #include <cmath>
00014 #include <thread>
00015
00016 #include "minimp3/minimp3.h"
00017 #include "minimp3/minimp3_ex.h"
00018
00019 #ifndef DECODER_HPP
00020 #define DECODER_HPP
00021
00022 constexpr float DEFAULT_VOLUME = 0.5f;
00023 using name_t = std::string;
00024
00029 struct AudioData {
00030     int16_t* pcmData;
00031     size_t total_samples;
00032     size_t current_sample;
00033     int channels;
00034     float volume = DEFAULT_VOLUME;
00035 };
00036
00041 struct MetaData {
00042     name_t track_name;
00043     name_t artist;
00044     name_t album;
00045     uint32_t duration;
00046 };
00047
00052 struct TrackInfo {
00053     MetaData meta_data;
00054     AudioData data;
00055     uint32_t sample_rate;
00056 };
00057
00058 class GenericTrack;
00059 using track_ptr_t = std::shared_ptr<GenericTrack>;
00060
00065 class Decoder {
00066 public:
00072     static track_ptr_t decode_mp3(const name_t& track_name);
00073 private:
00079     static MetaData parse_id3v1(const name_t& file_name);
00080 };
00081
00086 class GenericTrack {
00087 public:
00088     virtual ~GenericTrack() = default;
00092     virtual TrackInfo getTrackInfo() const = 0;
00096     virtual AudioData& getAudioDataRef() = 0;
00097     virtual void setCurrentSample(const size_t& position) = 0;
00098 };
00099
00100 class MP3Track : public GenericTrack {
00101 public:
00102     explicit MP3Track(const MetaData& meta_data, const AudioData& data, const unsigned int
    sample_rate)
00103         : meta_data_(meta_data), audio_data_(data), sample_rate_(sample_rate) {}
```

```
00104     TrackInfo getTrackInfo() const override { return TrackInfo {meta_data_, audio_data_,
      sample_rate_};}
00105     AudioData& getAudioDataRef() override { return audio_data_; }
00106     void setCurrentSample(const size_t& position) override { audio_data_.current_sample = position; }
00107 private:
00108     MetaData meta_data_;
00109     AudioData audio_data_;
00110     unsigned int sample_rate_;
00111 };
00112
00113 class ErrorTrack : public GenericTrack {
00114 public:
00115     explicit ErrorTrack(const name_t& message)
00116         : error_message_(message) {}
00117     TrackInfo getTrackInfo() const override { return TrackInfo {error_message_, "", "",0,
      AudioData()};}
00118     AudioData& getAudioDataRef() override { return error_data_; }
00119     void setCurrentSample(const size_t& position) override {}
00120 private:
00121     AudioData error_data_ = AudioData();
00122     name_t error_message_;
00123 };
00124
00125
00126 #endif //DECODER_HPP
```

8.4 main.cpp

```
00001 #include <iostream>
00002
00003 #include "ui_controller.hpp"
00004
00005
00006 int main(int argc, char *argv[]) {
00007
00008     UiController controller;
00009     controller.beginRenderLoop();
00010
00011     return 0;
00012 }
```

8.5 player.cpp

```
00001 //
00002 // Created by Darek Rudiš on 04.02.2025.
00003 //
00004 #include "player.hpp"
00005
00006 Player::Player() : paused_(false)
00007 {
00008     if (audio_.getDeviceCount() == 0) {
00009         throw std::runtime_error("No audio devices found!");
00010     }
00011
00012     params_.deviceId = audio_.getDefaultOutputDevice();
00013     params_.nChannels = 1;
00014     params_.firstChannel = 0;
00015 }
00016
00017 void Player::stop_track()
00018 {
00019     stop_playback_ = true;
00020 }
00021
00022 bool Player::is_stopped() const
00023 {
00024     return stop_playback_;
00025 }
00026
00027
00028 bool Player::is_paused() const
00029 {
00030     return paused_;
00031 }
00032
00033
00034 void Player::pause_track()
00035 {
00036     paused_ = true;
00037 }
00038
```

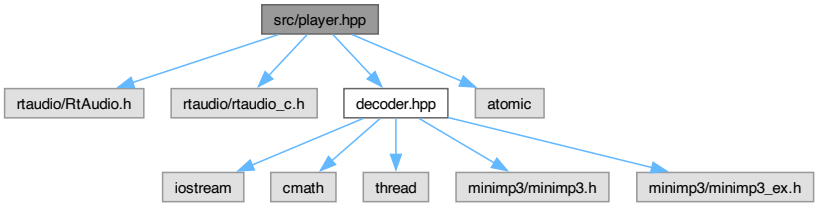
```
00039 void Player::resume_track()
00040 {
00041     if (paused_) {
00042         paused_ = false;
00043     }
00044 }
00045
00046 void Player::load_track(const track_ptr_t& track)
00047 {
00048     track_ = track;
00049     paused_ = false;
00050 }
00051
00052 void Player::raise_volume()
00053 {
00054     if (track_)
00055     {
00056         AudioData& audio_data = track_->getAudioDataRef();
00057         audio_data.volume = std::clamp(audio_data.volume + 0.02f, 0.0f, 1.0f);
00058         saved_volume_ = audio_data.volume;
00059     }
00060 }
00061 void Player::lower_volume()
00062 {
00063     if (track_)
00064     {
00065         AudioData& audio_data = track_->getAudioDataRef();
00066         audio_data.volume = std::clamp(audio_data.volume - 0.02f, 0.0f, 1.0f);
00067         saved_volume_ = audio_data.volume;
00068     }
00069 }
00070
00071
00072 void Player::play_track()
00073 {
00074     stop_playback_ = false;
00075
00076     if (track_ == nullptr)
00077     {
00078         throw std::runtime_error("No track found!");
00079     }
00080     if (audio_.isStreamOpen()) {
00081         std::cout « "Closing previous audio stream..." « std::endl;
00082         audio_.closeStream();
00083     }
00084
00085     TrackInfo info = track_->getTrackInfo();
00086     name_t name = info.meta_data.track_name;
00087     SAMPLE_RATE = info.sample_rate;
00088     params_.nChannels = info.data.channels;
00089     AudioData& audio_data = track_->getAudioDataRef();
00090     audio_data.volume = saved_volume_;
00091
00092     try {
00093         audio_.openStream(&params_, nullptr, RTAUDIO_SINT16, SAMPLE_RATE, &BUFFER_SIZE, audioCallback,
      &audio_data);
00094         audio_.startStream();
00095         while (audio_.isStreamRunning() && !stop_playback_) {
00096             if (audio_data.current_sample >= audio_data.total_samples) {
00097                 stop_playback_ = true;
00098             }
00099             if (is_paused())
00100                 audio_.stopStream();
00101
00102             while (is_paused() && !stop_playback_)
00103             {
00104                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
00105             }
00106
00107             if (!audio_.isStreamRunning() && !stop_playback_)
00108             {
00109                 audio_.startStream();
00110             }
00111             std::this_thread::sleep_for(std::chrono::milliseconds(100));
00112         }
00113         audio_.closeStream();
00114     } catch (RtAudioErrorType &e) {
00115         if (audio_.isStreamOpen()) audio_.closeStream();
00116     }
00117 }
00118     // Audio callback function
00119 int Player::audioCallback(void *outputBuffer, void *, unsigned int nFrames,
00120                   double, RtAudioStreamStatus, void *userData) {
00121     auto *audio_data = static_cast<AudioData *>(userData);
00122     if(!audio_data) return 0;
00123
00124     int16_t *buffer = static_cast<int16_t *>(outputBuffer);
```

```
00125     size_t channels = audio_data->channels;
00126     float volume = audio_data->volume;
00127
00128     for (unsigned int i = 0; i < nFrames; i++) {
00129         for (size_t ch = 0; ch < channels; ch++) {
00130             if (audio_data->current_sample < audio_data->total_samples) {
00131                 buffer[i * channels + ch] = static_cast<int16_t>(
00132                     audio_data->pcmData[audio_data->current_sample++] * volume);
00133             } else {
00134                 buffer[i * channels + ch] = 0;
00135             }
00136         }
00137     }
00138     return 0;
00139 }
```

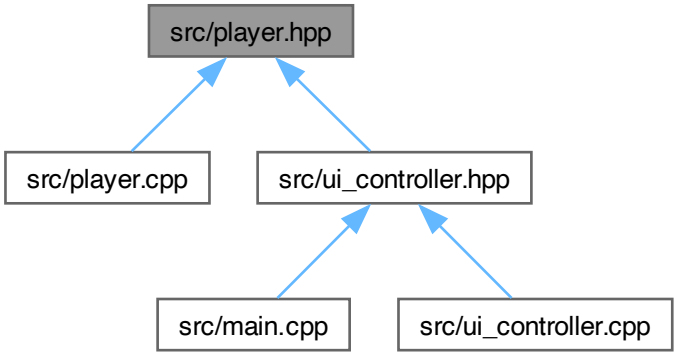
8.6 src/player.hpp File Reference

```
#include "rtaudio/RtAudio.h"
#include "rtaudio/rtaudio_c.h"
#include "decoder.hpp"
#include <atomic>
```

Include dependency graph for player.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Player

  Manages playback of data from a Track instance.

8.6.1 Detailed Description

Author

Darek Rudiš

Date

2025-02-10

Definition in file player.hpp.

8.7 player.hpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Darek Rudiš on 10.02.2025.
00003 //
00004
00010
00011 #include "rtaudio/RtAudio.h"
00012 #include "rtaudio/rtaudio_c.h"
00013 #include "decoder.hpp"
00014 #include <atomic>
00015
00016 #ifndef PLAYER_HPP
00017 #define PLAYER_HPP
00018
00029 class Player {
00030 public:
00034     explicit Player();
00035
00040     void load_track(const track_ptr_t& track);
00041
00052     void play_track();
00053     void raise_volume();
00054     void lower_volume();
00055     void pause_track();
00056     void resume_track();
00057     [[nodiscard]] bool is_paused() const;
00058
00062     void stop_track();
00063     bool is_stopped() const;
00064
00065 private:
00066     static int audioCallback(void *outputBuffer, void *, unsigned int nFrames,
00067                              double, RtAudioStreamStatus, void *userData);
00068     std::atomic<bool> stop_playback_{false};
00069     bool paused_;
00070     RtAudio audio_;
00071     RtAudio::StreamParameters params_;
00072     unsigned int BUFFER_SIZE = 512;
00073     unsigned int SAMPLE_RATE = 44100;
00074     float saved_volume_ = DEFAULT_VOLUME;
00075     track_ptr_t track_;
00076 };
00077
00078 #endif //PLAYER_HPP
```

8.8 ui_controller.cpp

```
00001 //
00002 // Created by Darek Rudiš on 25.02.2025.
00003 //
00004 #include "ui_controller.hpp"
00005
00006 UiController::UiController()
00007 {
00008     initscr();
00009     refresh();
00010     noecho();
00011     cbreak();
00012     keypad(stdscr, true);
00013     start_color();
00014     init_pair(1, COLOR_CYAN, COLOR_BLACK);
00015     curs_set(0);
00016     // Explicit initialization of the renderer after all setup ncurses functions have been called
00017     ui_ = UiRenderer();
```

```
00018 }
00019
00020 void UiController::beginRenderLoop()
00021 {
00022     path_ = std::filesystem::current_path().string();
00023
00024     updateFileList();
00025     nodelay(stdscr, TRUE);
00026     ui_.renderFileList(files_, highlight_);
00027     ui_.renderTrackQueue(track_queue_);
00028     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00029
00030     bool running = true;
00031     while (running)
00032     {
00033         ui_.updateAnimationFrame(current_track_, playing_, player_.is_paused());
00034         napms(50);
00035         if (!playing_ && !track_queue_.empty())
00036         {
00037             processNextTrackFromQueue();
00038         }
00039
00040         switch (int pressed_key = getch()) {
00041             case KEY_UP:
00042                 if (highlight_ > 0) highlight_--;
00043                 ui_.renderFileList(files_, highlight_);
00044                 break;
00045             case KEY_DOWN:
00046                 if (highlight_ < files_.size() - 1) highlight_++;
00047                 ui_.renderFileList(files_, highlight_);
00048                 break;
00049             case KEY_PLAY_TRACK:
00050                 processTrackSelection();
00051                 break;
00052             case KEY_ADD_QUEUE:
00053                 addTrackToQueue();
00054                 break;
00055             case KEY_PAUSE:
00056                 if (playing_) {
00057                     player_.is_paused() ? player_.resume_track() : player_.pause_track();
00058                 }
00059                 ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00060                 break;
00061             case KEY_NEXT_QUEUE:
00062                 processNextTrackFromQueue();
00063                 break;
00064             case KEY_VOLUME_UP:
00065                 player_.raise_volume();
00066                 break;
00067             case KEY_VOLUME_DOWN:
00068                 player_.lower_volume();
00069                 break;
00070             case KEY_QUIT:
00071                 stopTrackPlayback();
00072                 running = false;
00073                 break;
00074         }
00075     }
00076     endwin();
00077 }
00078
00079 void UiController::beginTrackPlayback(const track_ptr_t& track)
00080 {
00081     if (playback_thread_.joinable())
00082     {
00083         playback_thread_.join();
00084     }
00085     player_.load_track(track);
00086     playing_ = true;
00087     current_track_ = track;
00088     playback_thread_ = std::thread([this]() {
00089         player_.play_track();
00090         playing_ = false;
00091         current_track_ = nullptr;
00092         ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00093     });
00094 }
00095 void UiController::stopTrackPlayback()
00096 {
00097     if (playing_)
00098     {
00099         player_.stop_track();
00100         if (playback_thread_.joinable())
00101             playback_thread_.join();
00102         playing_ = false;
00103         current_track_ = nullptr;
00104     }
```

```
00105 }
00106
00107 void UiController::updateFileList()
00108 {
00109     files_.clear();
00110     files_.emplace_back(std::filesystem::directory_entry(std::filesystem::path(path_) / ".."));
00111     for (const auto& entry : std::filesystem::directory_iterator(path_))
00112     {
00113         files_.push_back(entry);
00114     }
00115 }
00116
00117 void UiController::showErrorPopup(const std::string &message) const
00118 {
00119     size_t height = 3;
00120     size_t width = message.size() + 4;
00121     size_t starty = (LINES - height) / 2;
00122     size_t startx = (COLS - width) / 2;
00123
00124     WINDOW* popup = newwin(height, width, starty, startx);
00125     box(popup, 0, 0);
00126     mvwprintw(popup, 1, 2, "%s", message.c_str());
00127     wrefresh(popup);
00128
00129     napms(1500);
00130
00131     delwin(popup);
00132     clear();
00133     refresh();
00134     ui_.renderFileList(files_, highlight_);
00135     ui_.renderTrackQueue(track_queue_);
00136     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00137 }
00138
00139 void UiController::processTrackSelection()
00140 {
00141     auto& selected = files_[highlight_];
00142     std::filesystem::path selected_path = selected.path();
00143
00144     if (selected_path.filename() == "..") {
00145         path_ = std::filesystem::path(path_).parent_path().string();
00146         highlight_ = 0;
00147         updateFileList();
00148     } else if (is_directory(selected_path)) {
00149         path_ = selected_path.string();
00150         highlight_ = 0;
00151         updateFileList();
00152     } else
00153     {
00154         if (playing_)
00155             stopTrackPlayback();
00156         track_ptr_t track = dec_.decode_mp3(selected_path.string());
00157         if (dynamic_cast<ErrorTrack*>(track.get()) != nullptr){
00158             //Check type of returned track
00159             showErrorPopup("Error opening file!");
00160             return;
00161         }
00162         beginTrackPlayback(track);
00163     }
00164     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
00165     ui_.renderFileList(files_, highlight_);
00166 }
00167
00168 void UiController::addTrackToQueue()
00169 {
00170     name_t track_path = files_[highlight_].path().string();
00171     track_ptr_t track = dec_.decode_mp3(track_path);
00172     if (dynamic_cast<ErrorTrack*>(track.get()) != nullptr) //Check type of returned track
00173     {
00174         showErrorPopup("Error opening file!");
00175         return;
00176     }
00177     track_queue_.emplace_back(track);
00178     ui_.renderTrackQueue(track_queue_);
00179 }
00180
00181 void UiController::processNextTrackFromQueue()
00182 {
00183     if (playing_)
00184         stopTrackPlayback();
00185     if (!track_queue_.empty())
00186     {
00187         auto next_track = track_queue_.front();
00188         track_queue_.pop_front();
00189         beginTrackPlayback(next_track);
00190     }
00191     ui_.renderStatusBar(current_track_, playing_, player_.is_paused());
```

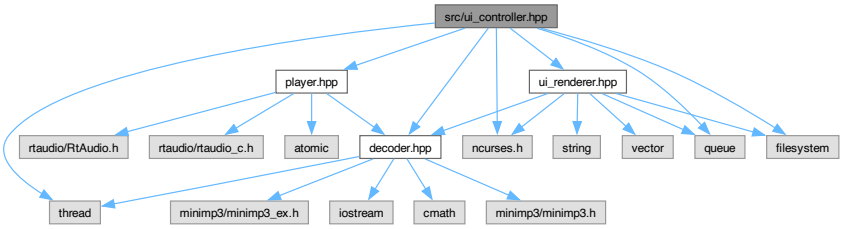
```
00192     ui_.renderTrackQueue(track_queue_);
00193 }
00194
00195
00196
00197
00198
```

8.9 src/ui_controller.hpp File Reference

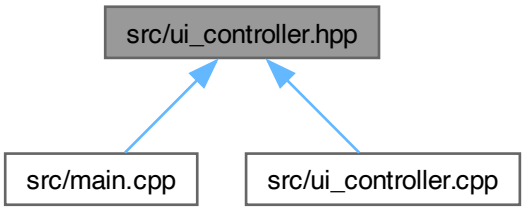
Defines class and constants regarding user interface core logic.

```
#include "decoder.hpp"
#include "player.hpp"
#include "ui_renderer.hpp"
#include <ncurses.h>
#include <queue>
#include <filesystem>
#include <thread>
```

Include dependency graph for ui_controller.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class UiController

  Takes care of user input handling and interaction between program components.

Typedefs

- using queue_t = std::deque<track_ptr_t>
- using file_t = std::filesystem::directory_entry
- using file_list_t = std::vector<file_t>

Variables

- constexpr char KEY_QUIT = 'q'
  Quit the application.
- constexpr char KEY_PAUSE = ' '
  Pause track playback.
- constexpr char KEY_NEXT_QUEUE = 'n'
  Skip current track and play first in queue.
- constexpr char KEY_ADD_QUEUE = 'a'
  Attempt to add selected file to queue.
- constexpr char KEY_PLAY_TRACK = '\n'
  General purpose "select" button.
- constexpr char KEY_VOLUME_UP = 'u'
  Raise playback volume.
- constexpr char KEY_VOLUME_DOWN = 'd'
  Lower playback volume.

8.9.1 Detailed Description

Defines class and constants regarding user interface core logic.

Author

Darek Rudiš

Date

2025-02-25

Definition in file ui_controller.hpp.

8.9.2 Typedef Documentation

8.9.2.1 file_list_t

```
using file_list_t = std::vector<file_t>
```

Definition at line 32 of file ui_controller.hpp.

8.9.2.2 file_t

```
using file_t = std::filesystem::directory_entry
```

Definition at line 31 of file ui_controller.hpp.

8.9.2.3 queue_t

```
using queue_t = std::deque<track_ptr_t>
```

Definition at line 30 of file ui_controller.hpp.

8.9.3 Variable Documentation

8.9.3.1 KEY_ADD_QUEUE

```
char KEY_ADD_QUEUE = 'a'  [constexpr]
```

Attempt to add selected file to queue.
Definition at line 25 of file ui_controller.hpp.

8.9.3.2 KEY_NEXT_QUEUE

```
char KEY_NEXT_QUEUE = 'n'  [constexpr]
```

Skip current track and play first in queue.
Definition at line 24 of file ui_controller.hpp.

8.9.3.3 KEY_PAUSE

```
char KEY_PAUSE = ' '  [constexpr]
```

Pause track playback.
Definition at line 23 of file ui_controller.hpp.

8.9.3.4 KEY_PLAY_TRACK

```
char KEY_PLAY_TRACK = '\n'  [constexpr]
```

General purpose "select" button.
Definition at line 26 of file ui_controller.hpp.

8.9.3.5 KEY_QUIT

```
char KEY_QUIT = 'q'  [constexpr]
```

Quit the application.
Definition at line 22 of file ui_controller.hpp.

8.9.3.6 KEY_VOLUME_DOWN

```
char KEY_VOLUME_DOWN = 'd'  [constexpr]
```

Lower playback volume.
Definition at line 28 of file ui_controller.hpp.

8.9.3.7 KEY_VOLUME_UP

```
char KEY_VOLUME_UP = 'u'  [constexpr]
```

Raise playback volume.
Definition at line 27 of file ui_controller.hpp.

8.10 ui_controller.hpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Darek Rudiš on 25.02.2025.
00003 //
00010
00011 #include "decoder.hpp"
00012 #include "player.hpp"
00013 #include "ui_renderer.hpp"
00014 #include <ncurses.h>
00015 #include <queue>
00016 #include <filesystem>
00017 #include <thread>
00018
00019 #ifndef UICONTROLLER_HPP
00020 #define UICONTROLLER_HPP
00021
00022 constexpr char KEY_QUIT = 'q';
00023 constexpr char KEY_PAUSE = ' ';
00024 constexpr char KEY_NEXT_QUEUE = 'n';
00025 constexpr char KEY_ADD_QUEUE = 'a';
00026 constexpr char KEY_PLAY_TRACK = '\n';
00027 constexpr char KEY_VOLUME_UP= 'u';
00028 constexpr char KEY_VOLUME_DOWN = 'd';
00029
00030 using queue_t = std::deque<track_ptr_t>;
00031 using file_t = std::filesystem::directory_entry;
00032 using file_list_t = std::vector<file_t>;
00033
00044 class UiController {
00045 public:
00049     UiController();
00050
00055     void beginRenderLoop();
00056 private:
00057     void beginTrackPlayback(const track_ptr_t& track);
00058     void stopTrackPlayback();
00059     void updateFileList();
00060     void showErrorPopup(const std::string& message) const;
00069     void processTrackSelection();
```

```
00070     void addTrackToQueue();
00071     void processNextTrackFromQueue();
00072
00073     Decoder dec_;
00074     Player player_;
00075     UiRenderer ui_;
00076     file_list_t files_;
00077     queue_t track_queue_;
00078     track_ptr_t current_track_;
00079     std::string path_;
00080     int highlight_ = 0;
00081     std::thread playback_thread_;
00082     bool playing_ = false;
00083 };
00084 #endif //UICONTROLLER_HPP
```

8.11 ui_renderer.cpp

```
00001 //
00002 // Created by Darek Rudiš on 01.04.2025.
00003 //
00004 #include "ui_renderer.hpp"
00005
00006 #include <__filesystem/operations.h>
00007
00008 UiRenderer::UiRenderer()
00009 {
00010     int mid_x = COLS / 2;
00011     file_list_win_ = newwin(LINES - 6, mid_x, 0, 0);
00012     track_queue_win_ = newwin(LINES - 6, COLS - mid_x, 0, mid_x);
00013     status_bar_win_ = newwin(6, COLS, LINES - 6, 0);
00014 }
00015
00016
00017 void UiRenderer::renderFileList(const file_list_t& files, size_t highlight) const
00018 {
00019     wclear(file_list_win_);
00020     box(file_list_win_, 0, 0);
00021     for (size_t i = 0; i < files.size(); ++i) {
00022         if (i == highlight)
00023             wattron(file_list_win_, A_REVERSE);
00024         if (std::filesystem::path(files[i]).extension() == ".mp3")
00025             wattron(file_list_win_, COLOR_PAIR(1));
00026
00027         if (is_directory(files[i]))
00028             mvwprintw(file_list_win_, i + 1, 2, "/%s", files[i].path().filename().c_str());
00029         else
00030             mvwprintw(file_list_win_, i + 1, 2, files[i].path().filename().c_str());
00031
00032         if (std::filesystem::path(files[i]).extension() == ".mp3")
00033             wattroff(file_list_win_, COLOR_PAIR(1));
00034         if (i == highlight)
00035             wattroff(file_list_win_, A_REVERSE);
00036     }
00037     wrefresh(file_list_win_);
00038 }
00039 void UiRenderer::renderTrackQueue(const std::deque<track_ptr_t> &queue) const
00040 {
00041     wclear(track_queue_win_);
00042     box(track_queue_win_, 0, 0);
00043     mvwprintw(track_queue_win_, 0, 2, "Track Queue");
00044     for(size_t i = 0; i < queue.size(); ++i)
00045     {
00046         mvwprintw(track_queue_win_, i + 1, 2, "%s",
      queue[i]->getTrackInfo().meta_data.track_name.c_str());
00047     }
00048     wrefresh(track_queue_win_);
00049 }
00050 void UiRenderer::renderStatusBar(const track_ptr_t& current_track, const bool& playing, const bool&
      paused) const
00051 {
00052     wclear(status_bar_win_);
00053     box(status_bar_win_, 0, 0);
00054     if (playing) {
00055         mvwprintw(status_bar_win_, 0, 2, "Now Playing: ");
00056         mvwprintw(status_bar_win_, 0, 15, "   ");
00057         if (paused)
00058         {
00059             wattron(status_bar_win_, A_BOLD);
00060             mvwprintw(status_bar_win_, 2, 2, "||");
00061         } else {
00062             wattron(status_bar_win_, A_BOLD);
00063             mvwprintw(status_bar_win_, 2, 2, "|>");
00064         }
```

```
00065         if (current_track != nullptr)
00066         {
00067             TrackInfo track_info = current_track->getTrackInfo();
00068             renderTrackPlayingText(track_info.meta_data);
00069         }
00070         wattroff(status_bar_win_, A_BOLD);
00071     }
00072     else
00073         mvwprintw(status_bar_win_, 0, 2, "Select a track and press ENTER to play");
00074     mvwprintw(status_bar_win_, 5, 2 , "[Q] Quit  [SPACE] Pause  [A] Add  [N] Next  [U/D] Volume
      Control");
00075     wrefresh(status_bar_win_);
00076 }
00077 void UiRenderer::renderTrackPlayingText(const MetaData& metaData) const
00078 {
00079     wattron(status_bar_win_, A_BOLD);
00080     mvwprintw(status_bar_win_, 2, 6, "%s", metaData.track_name.c_str());
00081     wattroff(status_bar_win_, A_BOLD);
00082     mvwprintw(status_bar_win_, 2, 6 + metaData.track_name.length(), " by: ");
00083     wattron(status_bar_win_, A_BOLD);
00084     mvwprintw(status_bar_win_, 2, 11 + metaData.track_name.length(), "%s", metaData.artist.c_str());
00085     wattroff(status_bar_win_, A_BOLD);
00086     mvwprintw(status_bar_win_, 2, 12 + metaData.track_name.length() + metaData.artist.length(), " on:
      ");
00087     wattron(status_bar_win_, A_BOLD);
00088     mvwprintw(status_bar_win_, 2, 17 + metaData.track_name.length() + metaData.artist.length(), "%s",
      metaData.album.c_str());
00089     wattroff(status_bar_win_, A_BOLD);
00090 }
00091 void UiRenderer::updateAnimationFrame(const track_ptr_t& current_track, const bool& playing, const
      bool& paused) const
00092 {
00093     static const char *frames[] = {
00094         ".!!!!.!!!!.", "!!!!!!.!!!!.", "!!!!!!.!!!..", "!!.!!!!.!!..", "!!..!!!!!.!.",
      "!....!!!..!!", "..!.!...!!.!", "..!!...!!!..", ".!!!...!!!!."
00095     };
00096     static size_t frame = 0;
00097     static size_t frame_counter = 0;
00098     if (playing)
00099     {
00100         if (!paused)
00101         {
00102             frame_counter++;
00103             if (frame_counter >= frame_delay) {
00104                 frame_counter = 0;
00105                 frame = (frame + 1) % std::size(frames);
00106             }
00107         }
00108         if (current_track != nullptr)
00109         {
00110             TrackInfo track_info = current_track->getTrackInfo();
00111             uint32_t elapsed_time_minutes = track_info.data.current_sample / track_info.sample_rate /
      track_info.data.channels / 60;
00112             uint32_t elapsed_time_seconds = track_info.data.current_sample / track_info.data.channels
      / track_info.sample_rate % 60;
00113             uint32_t total_minutes = track_info.meta_data.duration / 60;
00114             uint32_t total_seconds = track_info.meta_data.duration % 60;
00115             mvwprintw(status_bar_win_, 3, 6, "%d:%02d / %d:%02d",elapsed_time_minutes,
      elapsed_time_seconds, total_minutes, total_seconds);
00116         }
00117         mvwprintw(status_bar_win_, 0, 15, frames[frame]);
00118         wrefresh(status_bar_win_);
00119     }
00120 }
00121 void UiRenderer::refreshAll() const
00122 {
00123     wrefresh(file_list_win_);
00124     wrefresh(track_queue_win_);
00125     wrefresh(status_bar_win_);
00126 }
00127
00128
00129
00130
```

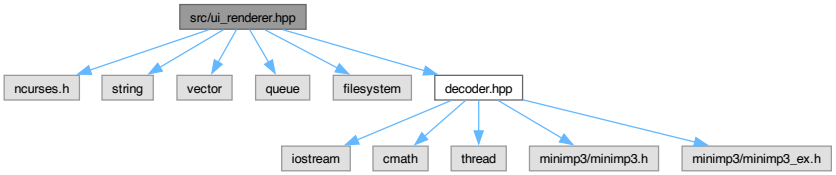
8.12 src/ui_renderer.hpp File Reference

Defines class handling UI window rendering specifics.

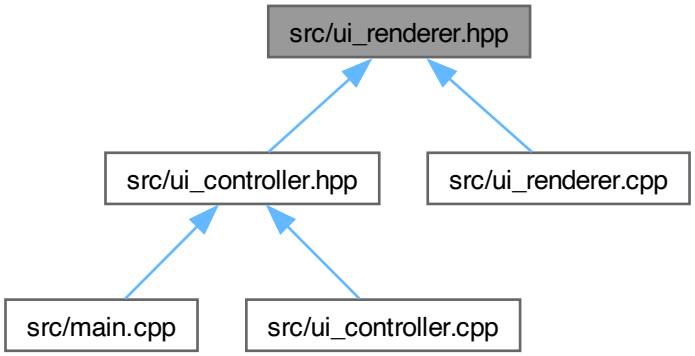
```
#include <ncurses.h>
#include <string>
#include <vector>
#include <queue>
```

```
#include <filesystem>
#include "decoder.hpp"
```

Include dependency graph for ui_renderer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class UiRenderer

  Renders updated UI state based on data received from controller class.

Typedefs

- using file_list_t = std::vector<file_t>

8.12.1 Detailed Description

Defines class handling UI window rendering specifics.

Author

Darek Rudiš

Date

2025-04-01

Definition in file ui_renderer.hpp.

8.12.2 Typedef Documentation

8.12.2.1 file_list_t

```
using file_list_t = std::vector<file_t>
```

Definition at line 21 of file ui_renderer.hpp.

8.13 ui_renderer.hpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Darek Rudiš on 01.04.2025.
00003 //
00010 #ifndef UI_RENDERER_HPP
00011 #define UI_RENDERER_HPP
00012
00013 #include <ncurses.h>
00014 #include <string>
00015 #include <vector>
00016 #include <queue>
00017 #include <filesystem>
00018 #include "decoder.hpp"
00019
00020 using file_t = std::filesystem::directory_entry;
00021 using file_list_t = std::vector<file_t>;
00022
00027 class UiRenderer {
00028 public:
00032     UiRenderer();
00033
00039     void renderFileList(const file_list_t& files, size_t highlight) const;
00044     void renderTrackQueue(const std::deque<track_ptr_t>& queue) const;
00045
00052     void renderStatusBar(const track_ptr_t& current_track, const bool& playing, const bool& paused)
      const;
00053     void refreshAll() const;
00054
00062     void updateAnimationFrame( const track_ptr_t& current_track,const bool& playing, const bool&
      paused) const;
00063
00064 private:
00065     void renderTrackPlayingText(const MetaData& meta_data) const;
00066     static constexpr int frame_delay = 5;
00068     WINDOW *file_list_win_;
00069     WINDOW *track_queue_win_;
00070     WINDOW *status_bar_win_;
00071 };
00072
00073 #endif //UI_RENDERER_HPP
```