**Міністерство освіти і науки України Національний університет «Львівська політехніка»**

**Кафедра ЕОМ**

**Курсова робота**

**з дисципліни:** «Системне програмне забезпечення»

**на тему:**

«Розробка програмного забезпечення для керування аудіофайлами. Розробка програми, що дозволяє керувати аудіофайлами та виконувати різні дії з їх обробкою на комп'ютері з операційною системою Windows.»

**Виконав:**

ст. гр. КІ-38

Ільків Данило

**Прийняв:**

Олексів М.В.

**Львів 2023**

1

**Завдання на курсовий проект**

**Завдання на курсовий проект:** “Розробка програмного забезпечення для відтворення аудіофайлів”.

**Мета роботи:** розробити програму для керування аудіофайлами. Програма призначена для виконання аудіофайлів з можливістю керування процесом відтворення.

Вимоги до курсового проекту:

1. Програма повинна працювати під управлінням Операційної

Системи *“Windows 10/11”*.

1. Програма повинна бути розроблена на мові програмування *C++* з використанням стандартної бібліотеки програмного інтерфейсу *“Windows”* та бібліотеки *“winmm.lib”* для роботи з драйвером звукової карти.
2. Програма повинна бути здатною відтворювати аудіофайли, які відповідають даним вимогам:
3. Частота дискретизації: *44100 Гц*;
4. Бітрейт: *16 біт*;
5. Аудиоканал: *Mono/Stereo*;
6. Програма повинна мати графічний інтерфейс користувача з таким функціоналом:
7. Робота з аудіофайлами формату *“.WAV”* та *“.MP3”*;
8. Завантажння одного, або декількох аудіофайлів;
9. Відтворення аудіофайлу в реальному часі, з відстеженням теперішньої позиції виконання;
10. Можливість задання позиції відтворення аудіофайлу;
11. Можливість задання параметрів гучності та швидкості аудіофайлу під час виконання;
12. Можливість керування станом процесу виконання: пауза, зациклювання, і т.д.;
13. Робота з метаданими файлу, які залежать від формату: назва композиції, автор композиції, тривалість композиції, і т.д.;

2

**АНОТАЦІЯ**

Розглянуто процес створення програми, що дозволяє здійснювати керування аудіофайлами на комп'ютері. У роботі описано основні етапи проектування програми, включаючи аналіз вимог, проектування архітектури та інтерфейсу користувача, розробку функціональності та тестування. В результаті була створена програма з інтуїтивним інтерфейсом, що дозволяє користувачеві здійснювати операції з аудіофайлами. Розроблене програмне забезпечення може бути корисним для користувачів, які працюють з великою аудіофайлами на комп'ютері з операційною системою *Windows*.

Програма написана на мові С++ з використанням стандартної бібліотеки програмного інтерфейсу *“Windows”* та бібліотеки *“winmm.lib”* для роботи з драйвером звукової карти. Кожен елемент має відповідний клас це дозволить в майбутньому розширювати функціонал і підвищувати продуктивність.

3

**ЗМІСТ**

[**ВСТУП.................................................................................................................................................4**](#_page_39_0)

**Розділ 1** [**Аналіз сучасних методів і засобів для розроблення програмного**](#_page_46_0)

[**забезпечення для аудіоплеєрів...............................................................................................................................5**](#_page_46_0)

[**Розділ 2 Розробка архітектури програми, алгоритмів її роботи та вибір засобів**](#_page_55_0)

[**програмування..............................................................................................................................12**](#_page_55_0)[**2.1. Обгрунтування розробки програми керування аудіофайлами...............................................................................................................................................12**](#_page_53_0)

[**2.2. Розробка структурної схеми програми .........................................................................................................................................................................14**](#_page_57_0)

[**2.3. Вибір засобів розробки ПЗ....................................................................................................................................................................16**](#_page_60_0)

[**Розділ 3 Програмування та реалізація розробленого рішення............................................................................................................................17**](#_page_73_0)

* 1. **Розробка модуля блоку взаємодії клієнта………………………………………………….18**

**3.1.1. Головне вікно програми……………………………………………………………………………………….17**

**3.2. Розробка модуля блоку взаємодії сервера………………………………………………….19**

**3.2.1. Клас AudioEngine……………………………………………………………………….19**

**3.2.2. Клас AudioSample……………………………………………………………………………………21**

**3.2.3. Клас PlayingAudio…………………………………………………………………………………....223.2.4. Приклад реалізації логіки відтворення аудіоданих**

**…………………………………………………………………………………………………...23**

**3.3. Розробка модуля взаємодії посередника між клієнтом та сервером……………………………………………………………………………………………..24**

**3.3.1. Клас AudioPlayer……………………………………………………………………….24**

**3.3.2. Клас MainAudioPlayer …………………………………………………………………………………………………...25**

**3.3.2.1. Методи для роботи з аудіофайлами**

**…………………………………………………………………………………………………...25**

**3.3.2.2. Методи для керування процесу виконання аудіофайлу**

**…………………………………………………………………………………………………...26**

[**Розділ 4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДЕМОНСТРАЦІЯ**](#_page_90_0)

[**РОБОТИ.........................................................................................................................................28**](#_page_90_0)

[**4.1. Тестування ПЗ.................................................................................................................................................................28**](#_page_105_0)

[**ВИСНОВКИ .............................................................................................................................................................30**](#_page_127_0)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ...........................................................................................................................................31**](#_page_129_0)

[**ДОДАТКИ.........................................................................................................................................32**](#_page_131_0)

[**ДОДАТОК А......................................................................................................................................................32**](#_page_131_0)

[**ДОДАТОК Б.........................................................................................................................................................78**](#_page_157_0)

4

**ВСТУП**

Розробка програми для роботи з аудіоданими та відтворення аудіофайлів на комп'ютері з операційною системою *Windows* є актуальною задачею у сфері програмування.

Основна мета проекту - створення високорівневої бібліотеки роботи з аудіоданими, їх відтворення, використовуючи програмний інтерфейс *“Windows”* та стандартну бібліотеку *“winmm.lib”* для роботи з драйвером звукової карти. Для демонстрації можливостей високорівневої бібліотеки, буде розроблено зручний та функціоний інтерфейс, який дозволить користувачеві виконувати контроль відтворення аудіофайлів в реальному часі.

Дане програмне забезпечення призначено для роботи з аудіоданими, а також для відтворення аудіофайлів в операційній системі Windows.

У підсумку, розробка високорівневої бібліотеки для роботи з аудіоданими, їх відтворення на комп'ютері з операційною системою *Windows* є складним та важливим процесом, що потребує ретельного аналізу вимог, проектування архітектури, розробки функціоналу та тестування. В результаті вдалих рішень та правильної реалізації бібліотеки може стати

корисним та зручним інструментом для роботи з аудіоданими та їх відтворенням на комп'ютері з операційною системою *Windows*. Дане ПЗ підійде усім користувачам, які активно працюють із аудіо.

5

**РОЗДІЛ 1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ І ЗАСОБІВ ДЛЯ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АУДІОПЛЕЄРІВ**

Розробка аудіоплеєрів є актуальною і поширеною задачею в галузі програмного забезпечення. Аудіоплеєр - це програма, призначена для відтворення аудіофайлів різних форматів і надання користувачу зручного інтерфейсу для керування відтворенням.

При розробці аудіоплеєру є кілька моментів, на які варто звернути увагу, зокрема:

Використання мови програмування та фреймворків: Для розробки аудіоплеєра можна використовувати різні мови програмування, такі як *C++*, *Java*, *Python*, тощо.

1. Користувацький інтерфейс. Користувач звертатиме свою увагу на реалізацію графічного інтерфейсу, тому варто обрати рішення з використанням графічних бібліотек (наприклад, *Win32API*, *Windows* *Forms*, *QT*, тощо).

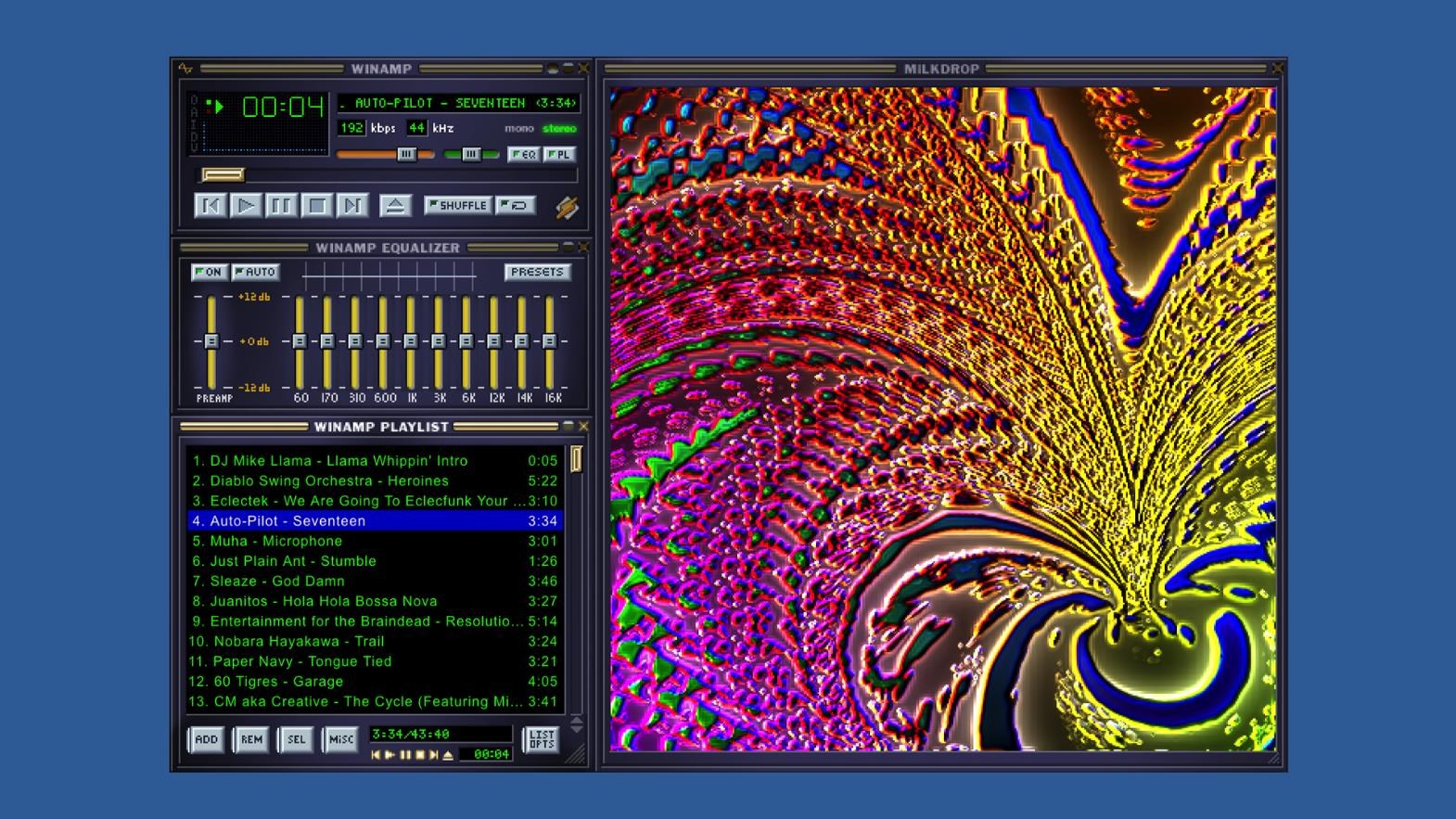
2. Функціональні можливості. Ключовою перевагою одного аудіоплеєра над іншим є наявність додаткового функціоналу. Це може бути як звичайне відтворення аудіофайлів різного формату, так і додаткові можливості у вигляді еквалайзера, фільтрів, відтворення онлайн-трансляцій, відтворення радіо, тощо.

3. Основна мета розробки програмного забезпечення полягає в забезпеченні користувачеві зручного інтерфейсу для моніторингу та аналізу системних показників.

Існує багато різних аудіоплеєрів, які мають свої особливості і функціональні можливості. Зокрема можна виділити наступні продукти:

6

***Winamp*** - один з найвідоміших аудіоплеєрів, який вже десятиліттями зберігає свою популярність. Winamp підтримує широкий спектр аудіоформатів і має різні функціональні можливості, включаючи створення плейлистів, налаштування еквалайзера, візуалізацію звуку тощо.



**Рис.1.1.** Вигляд аудіоплеєру ***Winamp*** у версії для *Windows 10/11*

Вочевидь можна виділити нестандартний дизайн та перелік можливостей, що в

теперішніх реаліях цілком відповідає потребам користувача.

7

***AIMP*** - безкоштовний аудіопрогравач з закритим початковим кодом, що займає

ключову нішу серед продуктів-конкурентів. Великий перелік підтримуваних аудіо-

форматів, еквалайзер, вбудовані звукові ефекти, робота одразу з кількома плейлистами, створення закладок, черг відтворення тощо.

**Рис.1.2.** Вигляд аудіоплеєру ***AIMP***



8

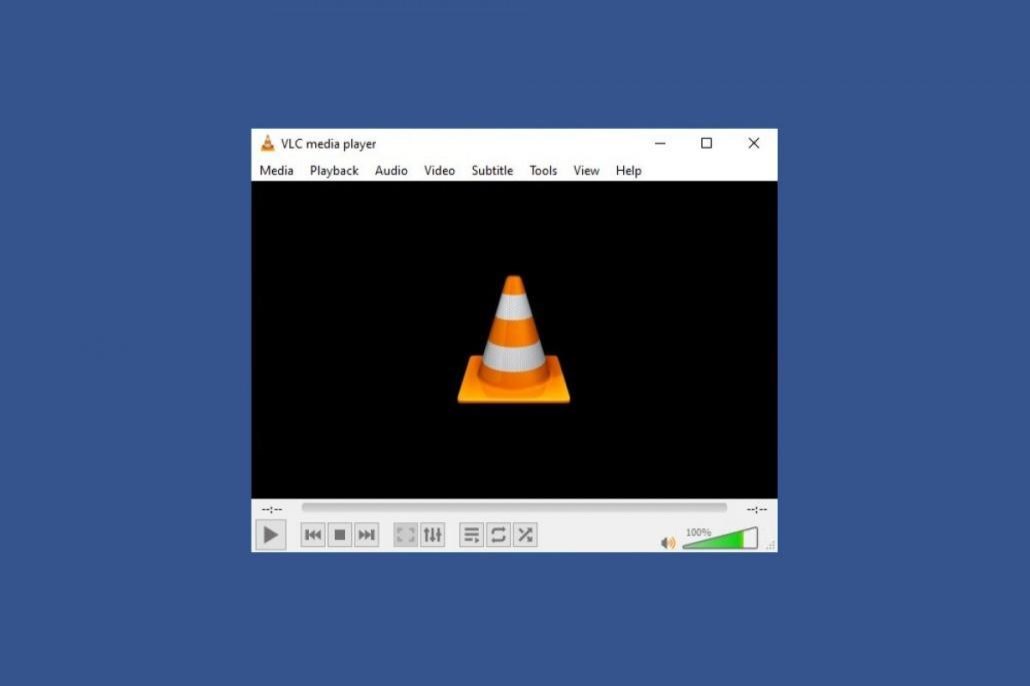
***iTunes*** - програма-медіаплеєр, яка застосовується для завантаження, зберігання, прослуховування та впорядкування файлів. Спочатку продукт подавався як **“.MP3”** програвач, однак з часом розширений функціонал почав включати відтворення відеофайлів також. Окрім базового функціоналу користувач має можливість редагування метаданих пісень, запису та імпорту композицій з CD дисків, багатосмуговий еквалайзер, візуалізатор, режим міні-плеєра тощо. Програма розроблена *Apple* та має пряму синхронізацію з їх продуктами, а також придбання композицій у фірмовому онлайн-магазині.

**Рис.1.3.** Вигляд аудіоплеєру ***iTunes***

9

***VLC Media Player*** - безкоштовний та відкритий аудіоплеєр, який підтримує відтворення аудіофайлів різних форматів. ***VLC*** має простий інтерфейс, але в той же час має розширений набір функцій, таких як налаштування еквалайзера, підтримка субтитрів, потокове відтворення медіа тощо.

**Рис.1.4.** Вигляд аудіоплеєру ***VLC***

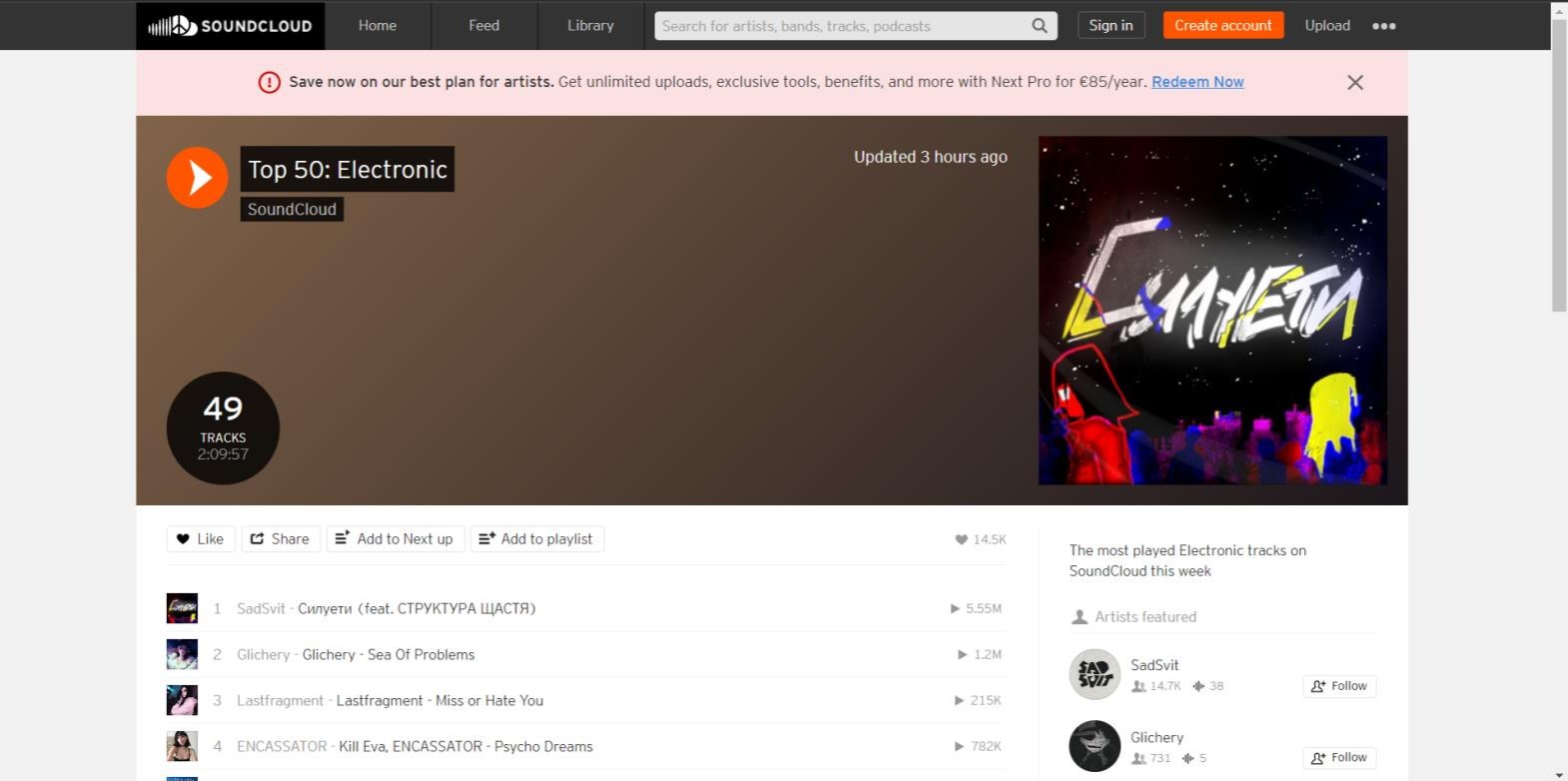


10

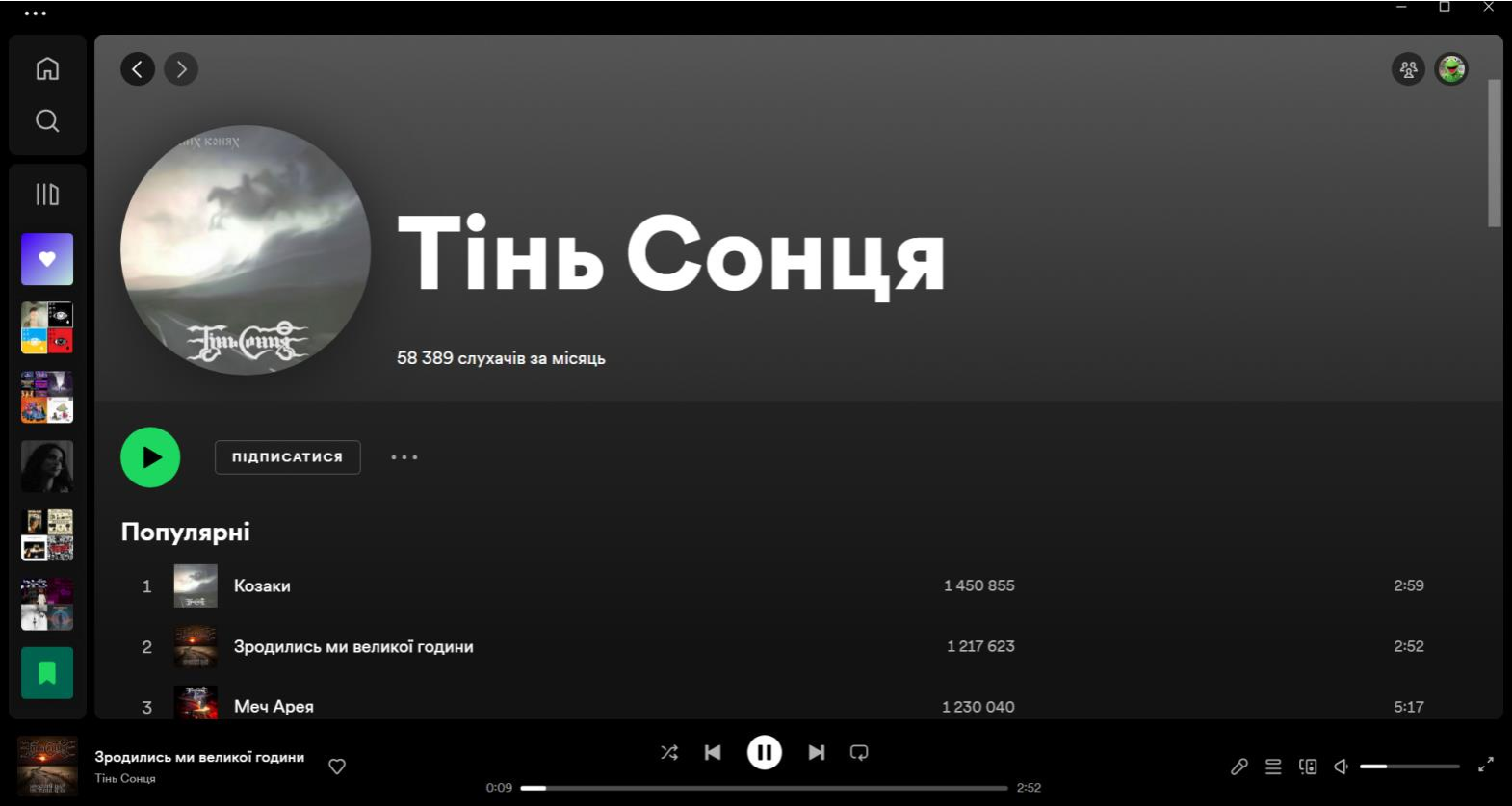
Однак варто звернути увагу, що користувачі в 2023 році надають перевагу аудіостріминговим сервісам, таким як ***SoundCloud***, ***Spotify***, ***Youtube Music*** тощо.

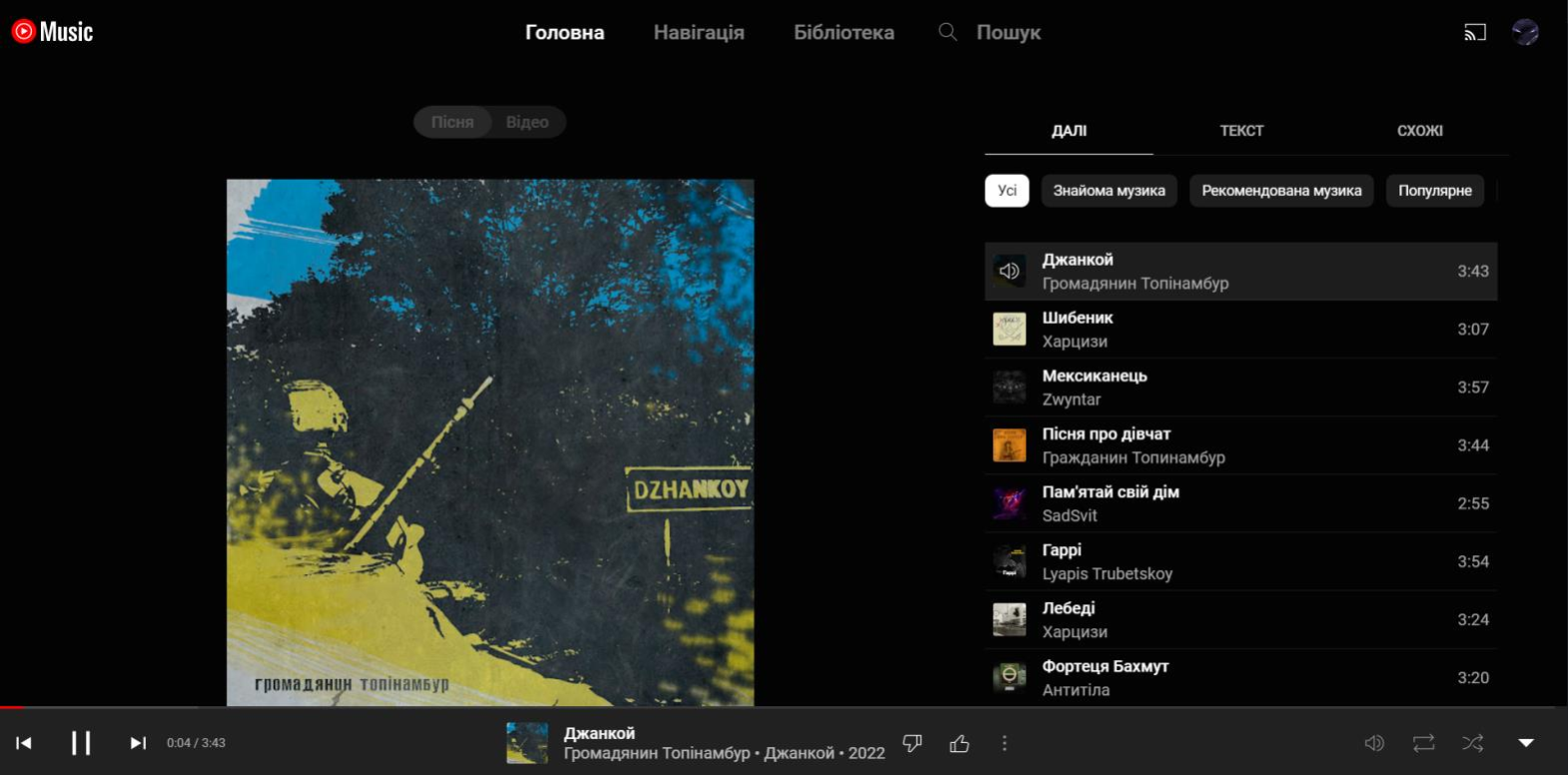
Пов’язано це через зручність використання, легку синхронізацію з сторонніми сервісами, відсутність необхідності локально зберігати аудіофайли та інтеграцію на мобільних пристроях.

**Рис. 1. 5.** Вигляд аудіострімингового сервісу ***Soundcloud***



11

**Рис. 1. 6.** Вигляд аудіострімингового сервісу ***Spotify***

**Рис. 1. 7.** Вигляд аудіострімингового сервісу ***Youtube Music***

Розробка аудіострімингових сервісів вимагає більше технічних можливостей, однак очевидно, що дані продукти вже цілком витісняють аудіоплеєри з ринку.

12

**РОЗДІЛ 2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ, АЛГОРИТМІВ ЇЇ РОБОТИ ТА ВИБІР ЗАСОБІВ ПРОГРАМУВАННЯ**

**2.1. Обгрунтування розробки програми керування файлової системи**

**Вимоги до програмного забезпечення** — набір вимог щодо властивостей, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться у розробці.

**Розробка програми має задовольняти такі вимоги:**

Ø**Надійність:**

ПЗ повинно належним чином реалізувати весь функціонал, який

описаний у документації, відображаючи високий рівень професійності та

уважності до деталей. Воно має забезпечувати стабільну та безперебійну

роботу, уникаючи взаємодії, які можуть вплинути на операційну систему або

призвести до втрати файлів. Робота з ПЗ повинна бути приємною та зручною

для користувачів, а його взаємодія з операційною системою має бути

безпечною та відповідати найвищим стандартам якості.

Ø**Керованість:**

Інтерфейс користувача ПЗ має бути простим та зручним для

використання незалежно від рівня технічної грамотності користувачів.

Ø**Швидкодія:**

Для максимально ефективної роботи з користувачем, необхідно, щоб

програмне забезпечення працювало швидко та надійно, уникаючи зайвих затримок.

13

Ø**Документація**:

Документація до програмного забезпечення має бути належним чином

підготовлена, щоб надати користувачам можливість правильно

використовувати його та зрозуміти всі його функції та можливості.

Ø**Покращення**:

ПЗ повинно бути гнучким щодо можливості майбутньої модифікації

шляхом додавання нових модулів або компонентів, щоб забезпечити його

адаптацію до змінних потреб користувачів.

14

**2.2. Розробка структурної схеми програми**

Структурна схема програми складається з трьох основних блоків:

1. Блок взаємодії *клієнта*

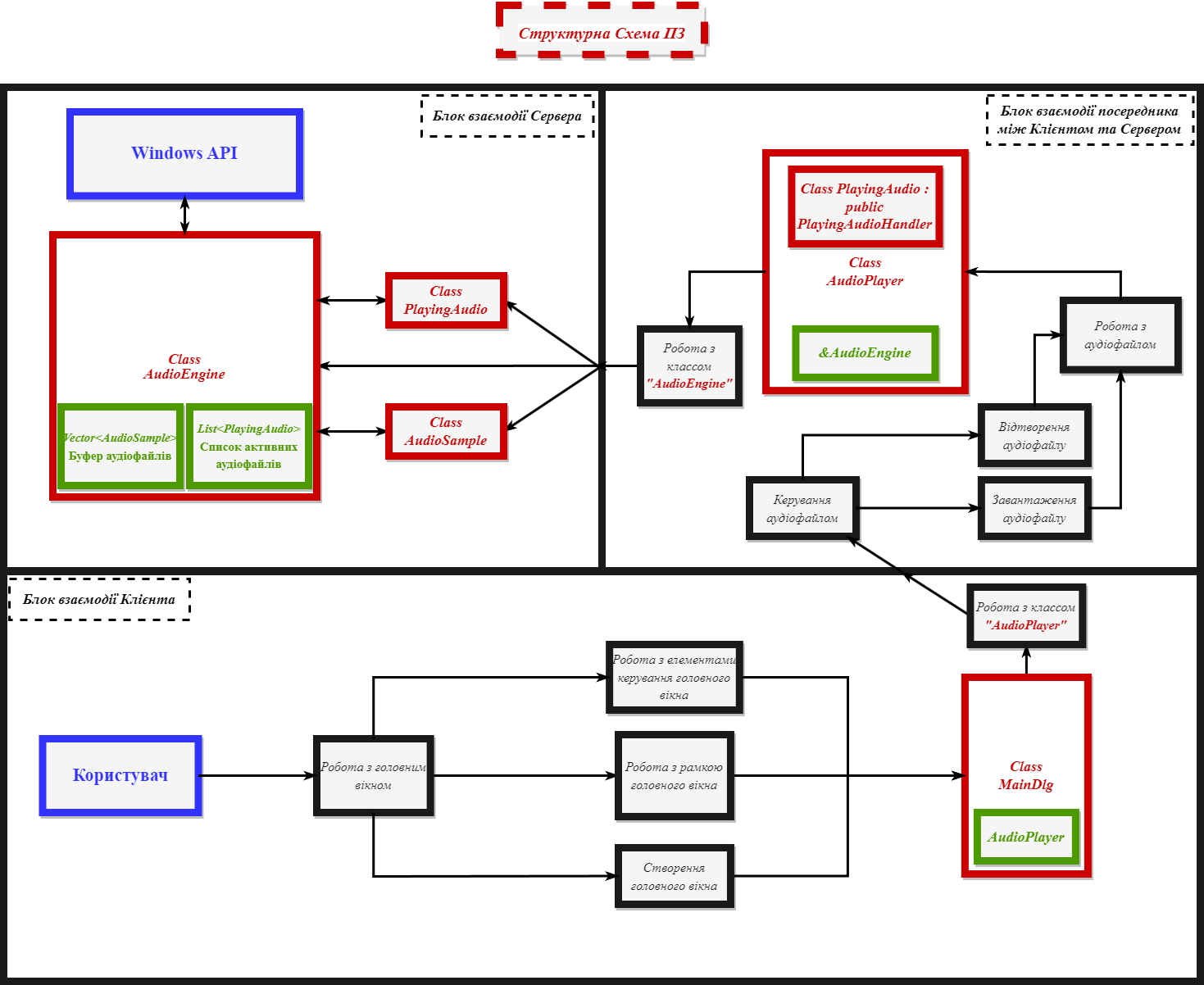
* *графічний інтерфейс* користувача;

1. Блок взаємодії *сервера*

* *аудіо рушій;*

1. Блок взаємодії *посередника* між *клієнтом* та *сервером*

* *аудіо програвач;*



**Рис 2.1.** Структурна схема ПЗ

15

***Блок взаємодії клієнта*** – це блок, який відповідає за графічний інтерфейс користувача. Він потрібен для отримання доступу до функціоналу ПЗ з сторони користувача.

Даний блок можна розділити на такі частини:

* 1. *головне вікно* – програмне вікно, що розташовується на робочому столі;
  2. *користувацький інтерфейс* – набір елементів керування;

***Блок взаємодії сервера*** – це блок, який відповідає за функціональну частину програми. Він потрібен для реалізації логіки ПЗ.

Даний блок можна розділити на такі частини:

1. *аудіо рушій* – реалізує роботу з драйвером *звукової карти*;
2. *аудіо зразок* – реалізує роботу з *аудіоданими;*
3. *аудіо мікшер* – реалізує відтворення *аудіоданих*;

***Блок взаємодії посередника між клієнтом та сервером*** – це блок, який відповідає за зв’язок між *клієнтом* та *сервером.* Він потрібен для організації зв’язку між графічним інтерфейсом та функціональною частиною ПЗ. Для цього він реалізує логіку роботи *сервера*, надаючи *клієнту* програмний інтерфейс для роботи з *аудіофайлами* та *аудіоданими*.

16

**2.3. Вибір засобів розробки ПЗ**

Для розробки Аудіо Рушія програми було обрано мову програмування *C++* та стандартну бібліотеку ОС *“Windows”* - *“winmm.lib”* для роботи з драйвером звукової карти.

Для розробки графічного інтерфейсу користувача було обрано стандартні бібліотеки програмного інтерфейсу ОС *“Windows”* - *“windows.h”*, а саме бібліотеки *“winuser.h”* та *“commctrl.h”*.

Також було використано редактор діалогового вікна середовища розробки *“Visual Studio”*.

Проект був розроблений у середовищі *Visual Studio*. Середовище має багато переваг:

- *Visual Studio* має широкий набір інструментів, які допомагають в розробці, відладці та тестуванні програм.

- Підтримка мови програмування *С++* з використанням бібліотеки програмного інтерфейсу ОС *“Windows”* - *Visual Studio* маєширокий набір інструментів, які допомагають в розробці програмного забезпечення.

- *Visual Studio* має потужні засоби для відлагодження програм.

Також, *Visual Studio* підтримує автоматичне тестування, що допомагає переконатися в якості та працездатності програми.

2.2. Опис використаних бібліотек

В проекті використовувалися такі бібліотеки:

1. *Windows API* – це загальна назва набору базових функцій інтерфейсів програмування програм операційних систем сімейств *Microsoft Windows* корпорації Майкрософт. Надає прямий спосіб взаємодії програм користувача з операційною системою Windows.
2. Стандартна бібліотека мультимедіа ОС *“Windows”*, *“mmsystem.h”*.

17

**РОЗДІЛ 3 ПРОГРАМУВАННЯ ТА РЕАЛІЗАЦІЯ РОЗРОБЛЕНОГО РІШЕННЯ**

Програма “AudioPlayer” складається з трьох основних частин.

**3.1. Розробка модуля блоку взаємодії клієнта**

У даному проекті було розроблено *7 класів*:

1. *BaseDlgBox* – відповідає за створення та роботу *діалогового вікна*.

Цей клас є абстрактним, для отримання доступу до програмного інтерфейсу, необхідно наслідувати та оприділити методи:

1. boolOnUserCreate(void);
2. bool OnUserDestroy(void);
3. LRESULT CALLBACK WindowProc(HWND \_In\_ hWnd,

UINT \_In\_ uMsg, WPARAM \_In\_ wParam, LPARAM \_In\_ lParam)

1. *MainDlg* – відповідає за створення та роботу *головного вікна* програми. Цей клас є нащадком класу *BaseDlgBox*.

Для роботи з даним класом необхідно створити екземпляр даного класу, передавши в конструктор параметри:

1. **dlgResName** - *ідентифікатор елемента ресурсу* що вказує на форму вікна, яке буде використовуватись *головним вікном*. Форма *головного вікна* створюється через редактор форми середовища *“Visual Studio”*.
2. **wcWavFile** - *ім'я файлу*, який був переданий через аргументи командного рядка.
3. **refAE** - *екземпляр аудіо рушія* з яким працює програма.
4. *Control* – абстрактний клас-інтерфейс. Використовується для реалізації класів з уніфікованим програмним інтерфейсом. Відповідає за створення та роботу елемента керування користувацького інтерфейсу.
5. *ControlButton* – відповідає за створення та роботу *кнопки*.
6. *ControlStaticText* – відповідає за створення та роботу *статичного тексту*.
7. *ControlCombobox* – відповідає за створення та роботу *викидного списку*.
8. *ControlSlider* – відповідає за створення та роботу *бігунка*.

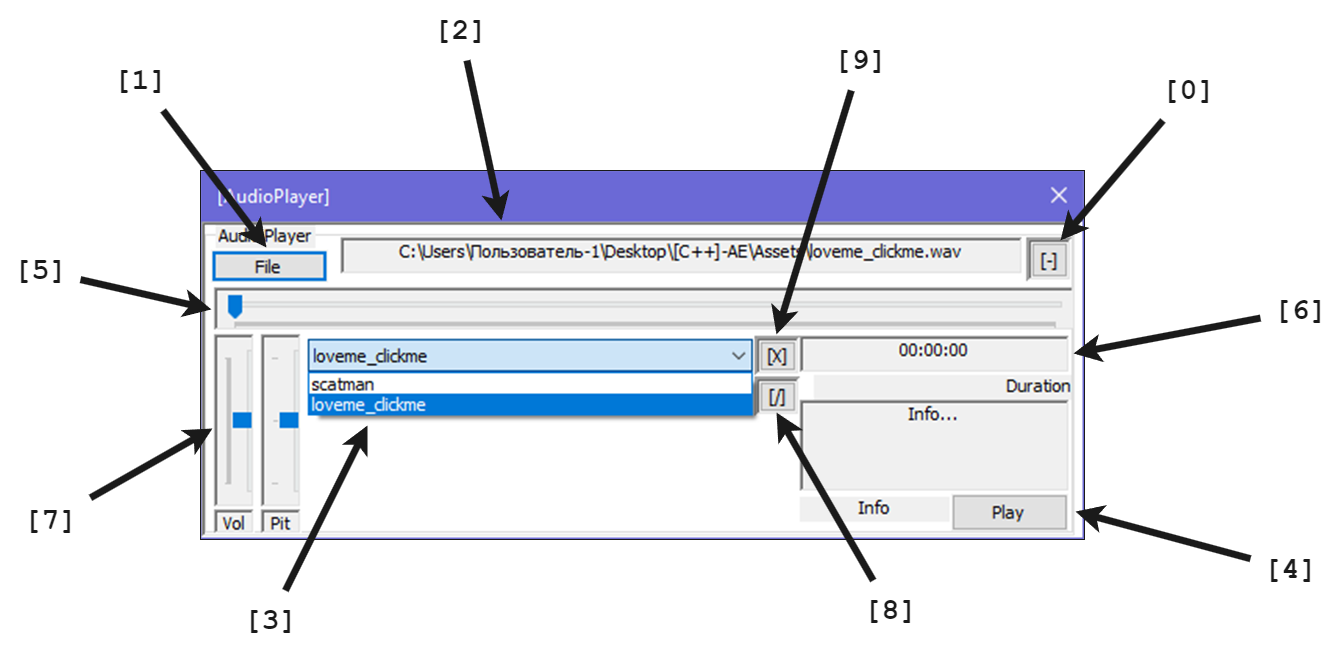
18

* + 1. **Головне вікно програми**

*Головне вікно* програми використовується для взаємодії з функціоналом ПЗ. На *головному вікні* відображається:

* 1. *Кнопка* згортання *головного вікна*. Дана кнопка зображена на **рис 3.1** під номером **0**.
  2. *Кнопка* вибору аудіокомпозиції. Використовується для вибору файлу з форматом *.WAV* або *.MP3*, що зберігається на компьютері. Дана кнопка зображена на **рис 3.1** під номером **1**.
  3. Шлях до файлу обраної композиції. Він зображений на **рис 3.1** під номером **2**.
  4. *Викидний список* аудіокомпозицій(*плейлист*). Після вибору аудіокомпозиції, вона потрапляє у *викидний список*(*плейлист*). Даний викидний список зображений на **рис 3.1** під номером **3**.
  5. *Кнопка* зміни стану виконання аудіокомпозиції. Використовується для *початку*/*продовження* відтворення або для *паузи*, залежно від стану виконання. Дана кнопка зображена на **рис 3.1** під номером **4**.
  6. *Бігунок* зміни позиції виконання аудіокомпозиції. Використовується для прогортання позиції виконання аудіокомпозиції в реальному часі. Даний бігунок зображений на **рис 3.1** під номером **5**.
  7. Теперішня позиція виконання в секундах/хвилинах/годинах (сек:хв:год). Віна зображена на **рис 3.1** під номером **6**.
  8. *Бігунки* гучності та швидкості виконання аудіокомпозиції. Використовуються для зміни *гучності* та *швидкості* виконання аудіокомпозиції в реальному часі. Дані бігунки зображені на **рис 3.1** під номером **7**.
  9. *Кнопка* зміни режиму виконання аудіокомпозиції (звичайне виконання; безперервне виконання; почергове виконання всіх композицій, що занесені до плейлисту). Дана кнопка зображена на **рис 3.1** під номером **8**.
  10. *Кнопка* скидання вибраної композиції. Дана кнопка зображена на **рис 3.1** під номером **9**.

19



**Рис 3.1.** Інтерфейс *головного вікна*

**3.2. Розробка модуля блока взаємодії сервера**

У даному проекті було розроблено *3 класи*:

* + 1. **Клас AudioEngine**

*AudioEngine* – відповідає за роботу з *драйвером звукової карти* та роботу з *буфером блоку аудіоданих*.

Цей клас працює з *драйвером звукової карти*, використовуючи програмний інтерфейс ОС *“Windows”*, а саме бібліотеку *“winmm.lib”*.

Для роботи з даним класом необхідно створити екземпляр даного класу, після чого проініціалізувати драйвер звукової карти.

Ініціалізація драйвера звукової карти виконується, звернувшись до методу:

bool CreateAudio(

unsigned int nSampleRate = 44100, unsigned int nBitsPerSample = 16,

unsigned int nChannels = 0x1, unsigned int nBlocks = 0x8,

unsigned int nBlockSamples = 512

)

Необхідно передати параметри:

1. **nSampleRate** *(частота дискретизації або частота семплювання)* – це кількість обчислень з одиницею аудіоданих за одиницю часу (одну секунду), що виконуються при заповненні пам’яті блоку аудіоданих. В даному випадку, частота дискретизації відповідає 44100 ГЦ - стандартна частота дискретизаціїї для CD (компакт диск).

20

1. **nBitsPerSample** *(бітрейт)* – це кількість біт на одиницю аудіоданих. В даному випатку одиниця аудіоданих кодується 16 бітами, тобто 2 байтами.
2. **nChannels** *(кількість аудіоканалів)* – моно/стерео звук. В даному випадку використовуються монофонічні аудіодані (1 канал).
3. **nBlocks** *(кількість блоків пам’яті)* – це кількість блоків пам’яті, з якими працює *аудіо рушій*. В даному випадку *аудіо рушій* працює з 8 блоками аудіоданих.
4. **nBlockSamples** *(розмір блоку пам’яті)* – це кількісь аудіоданих, що зберігає один блок пам’яті. В даному випадку один блок пам’яті зберігає 512 одиниць аудіоданих.

В результаті, драйвер звукової карти буду проініціалізовано. Тоді *аудіо рушій* створить потік в якому буде виконувати роботу з *списком активних аудіофайлів*.

Після ініціалізації драйвера звукової карти, з’являється можливість завантажити *аудіодані* вказавши ім’я, або повний шлях до *аудіофайлу* з форматом .WAV або .MP3. Для цього необхідно звернутись до методу:

AUDIOID LoadAudioSample(const wchar\_t\* wcWavFile)

В результаті, вміст файлу, що був вказаний, буде завантажено в *буфер аудіоданих,* для подальшого використання.

Після виконання, метод поверне в якості результуючого значення *ідентифікаток аудіоданих*. Його можна використати, щоб відтворити аудіофайл, що був завантажений. Для цього необхідно звернутись до методу:

void PlayAudioSample(AUDIOID ID)

В результаті, *аудіо рушій* занесе аудіофайл до *списку активних аудіофайлів*, після цього файл почне відтворюватись в реальному часі. Після того, як файл завершить відтворення, *аудіо рушій* вилучить його з *списку активних аудіофайлів.*

Для того, щоб завершити виконання *аудіо рушія*, необхідно звернутись до методу:

bool DestroyAudio(void);

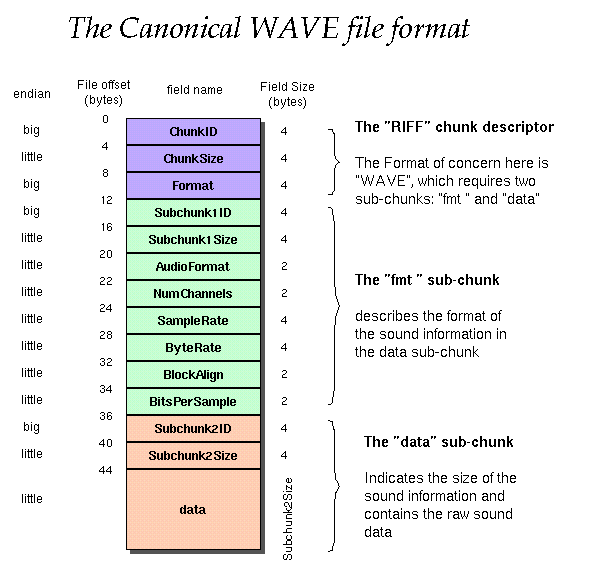
Після цього, потік, що був створений *аудіо рушієм* буде завершено.

21

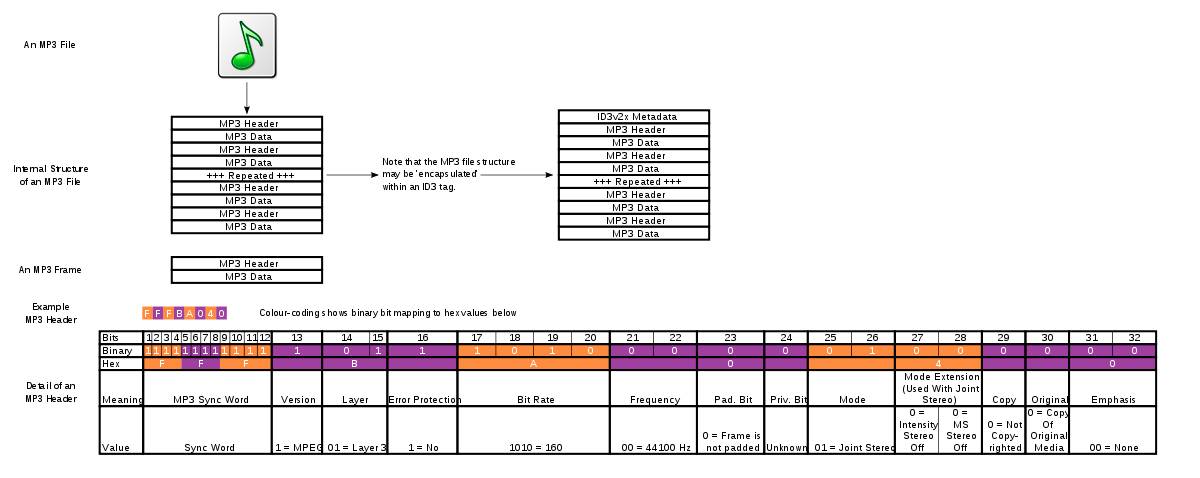
* + 1. **Клас AudioSample**

*AudioSample* – відповідає за зчитування та збереження аудіоданих з файлу. Для цього він покроково зчитує кожен байт аудіоданих та зберігає їх в пам’ять для подальшого використання.

Цей клас використовується *аудіо рушієм* для збереження аудіоданих в буфер для подальшого використання.



**Рис. 3.2.** Формат *.WAV* файлу



**Рис. 3.3.** Формат *.MP3* файлу

22

* + 1. **Клас PlayingAudio**

*PlayingAudio* – відповідає за відтворення аудіоданих, а також за перетворення аудіоданих при відтворенні.

Цей клас використовується *аудіо рушієм* для виконання аудіофайлів, що занесені до *списку активних аудіофайлів*.

За бажанням, цей клас можна наслідувати, для того, щоб реалізувати логіку відтворення аудіоданих. Для цього необхідно переоприділити метод:

float ProcessAudioSample(int nChannel,

float fGlobalTime, float fTimeStep, float fMixerSample,

const std::shared\_ptr<AudioEngine::AudioSample>& pS

)

Цей метод реалізує логіку відтворення аудіоданих. Він викликається *аудіо рушієм* при роботі з *списком активних аудіофайлів*. Під час звернення, *аудіо рушій* передає такі параметри:

1. **nChanel** *(канал)* – це номер аудіоканалу з яким працює *аудіо рушій* в даний момент часу.
2. **fGlobalTime** *(глобальний час)* – це загальна кількість часу, що пройшла з моменту початку роботи *аудіо рушія*.
3. **fTimeStep** *(крок часу)* – це кількість часу, яка необхідна для виконання обчислень з одиницею аудіоданих.
4. **fMixerSample** *(аудіодані мікшера)* – це аудіодані з якими працює *аудіо рушій* в даний момент часу.
5. **pS** *(a pointer to a sample)* – це вказівник на екземпляр аудіофайлу з яким працює *аудіо рушій* в даний момент часу.

23

* + 1. **Приклад реалізації логіки відтворення аудіоданих**

*Приклад реалізації:*

float PlayingAudio::ProcessAudioSample(int nChannel,

float fGlobalTime, float fTimeStep, float fMixerSample,

const std::shared\_ptr<AudioEngine::AudioSample>& pS

)

{

// Calculate sample position

m\_dSamplePosition.store(

m\_dSamplePosition.load() + (double)pS->wavHeader.nSamplesPerSec \* fTimeStep

);

// If sample position is valid add to the mix

if (m\_dSamplePosition < pS.get()->m\_nSamples)

{ fMixerSample += pS.get()->m\_fSample[((long)round(m\_dSamplePosition) \*

pS.get()->m\_nChannels) + nChannel];

}

else

{ m\_bFinish = true; } // Else sound has completed

return(fMixerSample);

}

Спочатку обчислюємо *лічильник* *позиції відтворення аудіоданих.*

// Calculate sample position

m\_dSamplePosition.store(

m\_dSamplePosition.load() + (double)pS->wavHeader.nSamplesPerSec \* fTimeStep

)

Тоді обчислюємо *значення аудіоданих* враховуючи теперішнє значення *лічильника* *позиції відтворення аудіоданих*. Після чого, повертаємо *результуюче значення аудіоданих*.

if (m\_dSamplePosition < pS.get()->m\_nSamples)

{ fMixerSample += pS.get()->m\_fSample[((long)round(m\_dSamplePosition) \*

pS.get()->m\_nChannels) + nChannel];

}

else

{ m\_bFinish = true; } // Else sound has completed

return(fMixerSample);

В даному прикладі аудіофайл буде відтворюватись до тих пір, поки *лічильник позиції відтворення* не досягне кінця аудіофайлу.

24

* 1. **Розробка модуля взаємодії посередника між клієнтом та сервером**

У даному проекті було розроблено *2 класи*:

* + 1. **Клас AudioPlayer**

*AudioPlayer* – відповідає за роботу з аудіофайлом, а також за реалізацію логіки відтворення аудіоданих.

Цей клас є абстрактним, для отримання доступу до програмного інтерфейсу, необхідно наслідувати та оприділити метод:

float AudioHandler(int nChannel,

float fGlobalTime, float fTimeStep, float fMixerSample,

const pAudioSample pS, const pPlayingAudio pA

)

Цей метод реалізує логіку відтворення аудіоданих. Він викликається класом *AudioPlayerHandler*. Цей клас реалізований всередині класу *AudioPlayer.* Під час звернення, *AudioPlayerHandler* передає такі параметри:

1. **nChanel** *(канал)* – це номер аудіоканалу з яким працює *аудіо рушій* в даний момент часу.
2. **fGlobalTime** *(глобальний час)* – це загальна кількість часу, що пройшла з моменту початку роботи *аудіо рушія*.
3. **fTimeStep** *(крок часу)* – це кількість часу, яка необхідна для виконання обчислень з одиницею аудіоданих.
4. **fMixerSample** *(аудіодані мікшера)* – це аудіодані з якими працює *аудіо рушій* в даний момент часу.
5. **pS** *(a pointer to a sample)* – це вказівник на екземпляр класу *AudioSample* з яким працює *аудіо рушій* в даний момент часу.
6. **pH** *(a pointer to a handler)* – це вказівник на екземпляр класу *AudioPlayerHandler* з яким працює *аудіо рушій* в даний момент часу.

*AudioPlayerHandler* – відповідає за виконання логіки відтворення аудіоданих. Цей клас є нащадком класу *PlayingAudio.*

25

* + 1. **Клас MainAudioPlayer**

*MainAudioPlayer* – відповідає за зв’язок між *клієнтом* та *сервером*, реалізує набір функцій для завантаження, виконання та керування процесом виконання аудіофайлу, а також реалізує логіку відтворення аудіоданих.

Для роботи з даним класом необхідно створити екземпляр даного класу, передавши в конструктор адресу екземпляру класу *AudioEngine*. Після цього з’являється можливість роботи з екземпляром даного класу.

У даному класі реалізовано *2 набори методів:*

* + - 1. **Методи для роботи з аудіофайлами**

Реалізовано метод для завантаження аудіофайла в *буфер аудіоданих* *аудіо рушія* для подальшого використання:

AUDIOID LoadAudioSample

(const wchar\_t\* wcWavFile)

**wcWavFile** *–* це *ім’я аудіофайлу* з форматом *.WAV.*

В результаті виконання методу, аудіофайл буде завантажено в *буфер аудіоданих* *аудіо рушія* для подальшого використання. Після завантаження, аудіофайл буде помічено як вибраний.

Після виконання, метод поверне якості результуючого значення *ідентифікатор аудіофайлу* для подальшого використання. Його можна використати, щоб вибрати аудіофайл, що був завантажений. Для цього необхідно звернутись до методу:

bool ChangeCurrentAudioSample

(AUDIOID ID)

В результаті, вказаний файл буде помічено як *вибраний*.

*Вибраний файл* – це файл з яким працює клас *MainAudioPlayer* в даний момент часу. Клас *MainAudioPlayer* може працювати тільки з *одним* аудіофайлом, проте, в *буфер аудіоданих* можна завантажити *довільну кількість* аудіофайлів.

Для того, щоб отримати *ідентифікатор поміченого аудіофайлу* необхідно звернутись до методу:

AUDIOID CurrentAudioSample(void) const

26

* + - 1. **Методи для керування процесом виконання аудіофайлу**

Реалізовано набір методів для керування *станом процесу виконання* аудіокомпозиції (*пауза*, *зациклювання*, і т.д.):

1. void ChangeStateAudio(signed int nState)
2. void SwapStateAudio(void)
3. signed int CurrentStateAudio(void) const
4. – метод зміни *стану* виконання. Приймає *ідентифікатор стану* виконання, та встановлює його як теперішній.
5. – метод перемикання *стану* виконання. Перемикає *стан* виконання на протилежний.
6. – метод отримання *ідентифікатора теперішнього стану* виконання. Повертає *ідентифікатор стану* виконання, який встановлений як теперішній.

Всього існує *3 можливих* *стани* виконання:

* STATE\_PLAY – аудіофайл починає/продовжує відтворення.
* STATE\_STOP – аудіофайл припиняє відтворення.
* STATE\_NULL – (не використовується).

Реалізовано набір методів для керування позицією відтворення аудіофайлу:

1. void PositonAudio(double dSamplePosition)
2. double CurrentPositonAudio(void) const
3. – метод зміни *позиції* відтворення. Приймає *числове значення* в діапазоні від 0 до 1, та встановлює його як значення теперішньої позиції відтворення.
4. – метод отримання *теперішньої позиції* відтворення. Повертає *числове значення* в діапазоні від 0 до 1.

27

Реалізовано набір методів для керування гучністю та швидкістю відтворення аудіофайлу:

1. void VolumeAudio(float fVolume)
2. float CurrentVolume(void) const
3. void PitchAudio(float fPitch)
4. float CurrentPitch(void) const
5. – метод зміни *гучності* відтворення. Приймає *числове значення* починаючи з 0, та встановлює його як значення теперішньої гучності відтворення.
6. – метод отримання *теперішньої гучності* відтворення. Повертає *числове значення* в починаючи з 0.
7. – метод зміни *швидкості* відтворення. Приймає *числове значення* починаючи з 0, та встановлює його як значення теперішньої швидкості відтворення.
8. – метод отримання *теперішньої швидкості* відтворення. Повертає *числове значення* в починаючи з 0.

Це основні класи та методи, які використовуються в ПЗ. Детальний опис, а також код проекту наведений в Додатку А, діаграму класів наведено в Додатку Б.

28

[**РОЗДІЛ 4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДЕМОНСТРАЦІЯ**](#_page_90_0)[**РОБОТИ**](#_page_90_0)

**4.1. Тестування ПЗ**

Тестування програмного забезпечення має велике значення в процесі розробки продукту, оскільки на цьому етапі виявляються помилки, які були зроблені на попередніх етапах. Цей етап дозволяє поліпшити різні аспекти продукту, наприклад, його інтерфейс, і знаходить слабкі місця, якщо такі є, для подальшого їх виправлення.

Після завершення реалізації розробки програмного забезпечення для відтворення аудіофайлів на комп'ютері з ОС “Windows”, можна зробити висновок, що всі поставлені задачі були виконані.

Поставлені задачі:

1. Робота з аудіофайлами формату *“.WAV”* та *“.MP3”*;
2. Завантажння одного, або декількох аудіофайлів;
3. Відтворення аудіофайлу в реальному часі, з відстеженням теперішньої позиції виконання;
4. Можливість задання позиції відтворення аудіофайлу;
5. Можливість задання параметрів гучності та швидкості аудіофайлу під час виконання;
6. Можливість керування станом процесу виконання: пауза, зациклювання, і т.д.;
7. Робота з метаданими файлу, які залежать від формату: назва композиції, автор композиції, тривалість композиції, і т.д.;

При тестуванні ПЗ було виявлено 1 баг:

При виконанні обчислення логіки відтворення аудіоданих відбувається переповнення розрядної сітки змінної, яка зберігає значення *лічильника позиції відтворення аудіоданих*, в результаті чого, значення різниці між теперішнім значенням *лічильника* та попереднім дорівнює 0 (аудіокомпозиція припиняє виконання).

29

Це стається тому що, розмір змінної є 4 байти, що є недостатньо для збереження великої кількості розрядів після коми, через що, значення заокруглюється до 0. Для виправлення цього, необхідно збільшити розмір змінної до 8 байт, тоді розмір змінної буде достатньо великий для проведення обчислень.

30

**ВИСНОВКИ**

Під час виконання курсового проекту було розроблено програму для керування аудіофайлами. Програма призначена для виконання аудіофайлів з можливістю керування процесом відтворення.

Програма має графічний інтерфейс користувача з таким функціоналом:

1. Робота з аудіофайлами формату *“.WAV”* та *“.MP3”*;
2. Завантажння одного, або декількох аудіофайлів;
3. Відтворення аудіофайлу в реальному часі, з відстеженням теперішньої позиції виконання;
4. Можливість задання позиції відтворення аудіофайлу;
5. Можливість задання параметрів гучності та швидкості аудіофайлу під час виконання;
6. Можливість керування станом процесу виконання: пауза, зациклювання, і т.д.;
7. Робота з метаданими файлу, які залежать від формату: назва композиції, автор композиції, тривалість композиції, і т.д.;

Програма написана на мові С++ з використанням стандартної бібліотеки програмного інтерфейсу *“Windows”* та бібліотеки *“winmm.lib”* для роботи з драйвером звукової карти.

Дане ПЗ підійде усім користувачам, які активно працюють із аудіо.

31

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. [MSDN] Windows Multimedia [Електронний ресурс]:

<https://learn.microsoft.com/en-us/windows/win32/Multimedia/windows-multimedia-start-page>

1. [StackOverflow/Questions] Reading the Data of a .WAV file [Електронний ресурс]: <https://stackoverflow.com/questions/13660777/c-reading-the-data-part-of-a-wav-file>
2. [StackOverflow/Questions] Reading the Data of a .MP3 file [Електронний ресурс]: <https://stackoverflow.com/questions/2968656/reading-mp3-files>
3. [VNS.LPNU]Системне програмне забезпечення МЕТОДИЧНІ ВКАЗІВКИ до виконання курсового проєкту для студентів базового напрямку “Комп’ютерна інженерія” [Електронний ресурс]: <https://vns.lpnu.ua/pluginfile.php?file=%2F3636211%2Fmod_resource%2Fcontent%2F1%2F%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BD%D1%96_%D0%B2%D0%BA%D0%B0%D0%B7%D1%96%D0%B2%D0%BA%D0%B8_%D0%BA%D1%83%D1%80%D1%81%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82_%D1%81%D0%BF%D0%B7.pdf>
4. Євген Музиченко. Низькорівневе програмування звуку в ОС “Windows”. Журнал Комп’ютер Пресс #6-2000.
5. Юрій Щупак. Win32 API. Розробка програм для Windows. СПб.: Питер, 2008. - 592 с.: ил.

32

**ДОДАТКИ**

**ДОДАТОК А**

*Лістинг коду з файлу* “main.cpp”

|  |
| --- |
| #include "framework.h" |
|  |
| #include "logger.h" |
|  |
| #include "maindlg.h" |
| #include "resource.h" |
|  |
| #include "audio\_engine.h" |
|  |
| using namespace std; |
|  |
| float MakeNoice(int nChannel, float fGlobalTime, float fTimeStep) { |
| float fOutput = 0.f; |
| fOutput = 0.5f \* sinf(440.f \* 3.14159f \* 2.f \* fGlobalTime); |
| return(0.f \* fOutput); |
| } |
|  |
| int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance, |
| \_In\_opt\_ HINSTANCE hPrevInstance, |
| \_In\_ LPWSTR lpCmdLine, \_In\_ int nCmdShow) |
| { |
| UNREFERENCED\_PARAMETER(hPrevInstance); |
|  |
| INITCOMMONCONTROLSEX icc = { |
| sizeof(icc), ICC\_WIN95\_CLASSES |
| }; |
| if (!InitCommonControlsEx(&icc)) |
| { Logger::ShowLastError |
| (L"InitCommonControlsEx(&icc)"); return(-0x1); |
| } |
|  |
| LPWSTR\* szArglist; |
| signed int nArgs; |
|  |
| szArglist = CommandLineToArgvW |
| (GetCommandLineW(), &nArgs); |
|  |
| wchar\_t\* wcWavFile = NULL; |
| if (szArglist == NULL) |
| { Logger::ShowLastError |
| (L"CommandLineToArgvW - Failed!"); return(-0x1); |
| } else if (nArgs > 0x1) |
| { wcWavFile = szArglist[0x1]; } |
|  |
| Logger::LoadLogLevel |
| (Logger::LogLevel::LOG\_LVL\_DEBUG); |
| Logger::ClearLog(); |
|  |
| AudioEngine audio(&MakeNoice); |
| audio.CreateAudio(); |
|  |
| MainDlg\* mainDlg = new |
| MainDlg(MAKEINTRESOURCE(IDD\_MAIN\_DIALOG), wcWavFile, audio); |
| mainDlg->CreateDlg(hInstance, NULL), mainDlg->ShowDlg(nCmdShow); |

33

|  |
| --- |
| MSG msg = { 0x0 }; |
| while (GetMessage(&msg, NULL, 0x0, 0x0) > 0x0) |
| if (!IsDialogMessage(mainDlg->GetDialogHWND(), &msg)) |
| { TranslateMessage(&msg), DispatchMessage(&msg); } |
|  |
| audio.DestroyAudio(); |
|  |
| LocalFree(szArglist); |
| return((int)msg.wParam); |
| } |

*Лістинг коду з файлу* “framework.h”

|  |
| --- |
| #ifndef \_FRAMEWORK\_H\_ |
| #define \_FRAMEWORK\_H\_ |
|  |
| #ifndef \_\_cplusplus |
| #error Error! :: Please use the "C++" |
| #endif |
|  |
| #pragma comment(linker,"\"/manifestdependency:type='win32' \ |
| name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \ |
| processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"" \ |
| ) |
|  |
| #pragma comment(lib, "winmm.lib") |
|  |
| #ifndef UNICODE |
| #error Error! :: Please enable UNICODE for your compiler! VS: Project Properties -> General -> \ |
| Character Set -> Use Unicode. Thanks! |
| #endif |
|  |
| #include "targetver.h" |
| // Исключите редко используемые компоненты из заголовков Windows |
| /\*#define WIN32\_LEAN\_AND\_MEAN\*/ |
| // Файлы заголовков Windows |
| #include <windows.h> |
| #include <windowsx.h> |
|  |
| #include <commctrl.h> |
| #pragma comment(lib, "comctl32.lib") |
|  |
| #include <psapi.h> |
| // Файлы заголовков среды выполнения C |
| #include <stdlib.h> |
| #include <malloc.h> |
| #include <memory.h> |
| #include <inttypes.h> |
| // Файлы заголовков среды выполнения C++ |
| #include <iostream> |
|  |
| #include <string> |
| #include <vector> |
| #include <list> |
|  |
| #include <memory> |
| #include <functional> |
| #include <chrono> |

34

|  |
| --- |
| #include <thread> |
| #include <atomic> |
| #include <condition\_variable> |
|  |
| #ifndef LOG\_FILE\_NAME |
| #define LOG\_FILE\_NAME L".AudioEngineLog.txt" |
| #endif |
|  |
| typedef signed int AUDIOID; |
| typedef std::function |
| <float(int, float, float)> AUDIO\_HANDLER; |
|  |
| #endif //\_FRAMEWORK\_H\_ |

*Лістинг коду з файлу* “targetver.h”

|  |
| --- |
| #pragma once |
|  |
| #include <SDKDDKVer.h> |

*Лістинг коду з файлу* “resource.h”

|  |
| --- |
| //{{NO\_DEPENDENCIES}} |
| // Включаемый файл, созданный в Microsoft Visual C++. |
| // Используется resource.rc |
| // |
| #define IDD\_MAIN\_DIALOG 101 |
| #define ID\_FILE 1001 |
| #define ID\_PLAY 1002 |
| #define IDC\_AUDIO\_TRACK 1003 |
| #define IDC\_STATIC\_0 1004 |
| #define ID\_X 1004 |
| #define IDC\_STATIC\_1 1005 |
| #define ID\_PB\_STATE 1005 |
| #define IDC\_DURATION 1006 |
| #define IDC\_STATIC\_3 1007 |
| #define IDC\_VOLUME 1008 |
| #define IDC\_FILE\_NAME 1009 |
| #define IDC\_STATIC\_2 1010 |
| #define IDC\_GROUPBOX\_0 1011 |
| #define IDC\_PITCH 1012 |
| #define IDC\_STATIC\_4 1013 |
| #define IDC\_INFO 1014 |
| #define IDC\_STATIC\_5 1015 |
| #define ID\_HIDE\_WINDOW 1016 |
| #define IDC\_PLAYLIST 1017 |
|  |
| // Next default values for new objects |
| // |
| #ifdef APSTUDIO\_INVOKED |
| #ifndef APSTUDIO\_READONLY\_SYMBOLS |
| #define \_APS\_NEXT\_RESOURCE\_VALUE 104 |
| #define \_APS\_NEXT\_COMMAND\_VALUE 40001 |
| #define \_APS\_NEXT\_CONTROL\_VALUE 1020 |
| #define \_APS\_NEXT\_SYMED\_VALUE 101 |
| #endif |
| #endif |

35

*Лістинг коду з файлу* “resource.rc”

|  |
| --- |
| // Microsoft Visual C++ generated resource script. |
| // |
| #include "resource.h" |
|  |
| #define APSTUDIO\_READONLY\_SYMBOLS |
| ///////////////////////////////////////////////////////////////////////////// |
| // |
| // Generated from the TEXTINCLUDE 2 resource. |
| // |
| #include "winres.h" |
|  |
| ///////////////////////////////////////////////////////////////////////////// |
| #undef APSTUDIO\_READONLY\_SYMBOLS |
|  |
| ///////////////////////////////////////////////////////////////////////////// |
| // Русский (Россия) resources |
|  |
| #if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_RUS) |
| LANGUAGE LANG\_RUSSIAN, SUBLANG\_DEFAULT |
| #pragma code\_page(1251) |
|  |
| #ifdef APSTUDIO\_INVOKED |
| ///////////////////////////////////////////////////////////////////////////// |
| // |
| // TEXTINCLUDE |
| // |
|  |
| 1 TEXTINCLUDE |
| BEGIN |
| "resource.h\0" |
| END |
|  |
| 2 TEXTINCLUDE |
| BEGIN |
| "#include ""winres.h""\r\n" |
| "\0" |
| END |
|  |
| 3 TEXTINCLUDE |
| BEGIN |
| "\r\n" |
| "\0" |
| END |
|  |
| #endif // APSTUDIO\_INVOKED |

36

|  |
| --- |
| ///////////////////////////////////////////////////////////////////////////// |
| // |
| // Dialog |
| // |
|  |
| IDD\_MAIN\_DIALOG DIALOGEX 0, 0, 376, 124 |
| STYLE DS\_ABSALIGN | DS\_SETFONT | DS\_MODALFRAME | DS\_FIXEDSYS | DS\_CENTER | WS\_CAPTION | WS\_SYSMENU |
| EXSTYLE WS\_EX\_OVERLAPPEDWINDOW |
| CAPTION "[AudioPlayer]" |
| FONT 8, "MS Shell Dlg", 400, 0, 0x1 |
| BEGIN |
| DEFPUSHBUTTON "File",ID\_FILE,4,11,50,13,NOT WS\_TABSTOP |
| PUSHBUTTON "Play",ID\_PLAY,320,107,50,15,NOT WS\_TABSTOP |
| CONTROL "",IDC\_AUDIO\_TRACK,"msctls\_trackbar32",TBS\_AUTOTICKS | TBS\_TOOLTIPS | WS\_BORDER,5,26,368,17,WS\_EX\_TRANSPARENT |
| GROUPBOX "Audio Player",IDC\_GROUPBOX\_0,0,0,373,124,0,WS\_EX\_TRANSPARENT | WS\_EX\_CLIENTEDGE |
| CTEXT "[...]",IDC\_DURATION,255,45,115,15,SS\_WORDELLIPSIS | WS\_BORDER,WS\_EX\_TRANSPARENT |
| CTEXT "Duration",IDC\_STATIC\_2,261,60,110,10,0,WS\_EX\_TRANSPARENT | WS\_EX\_RIGHT |
| CONTROL "",IDC\_VOLUME,"msctls\_trackbar32",TBS\_AUTOTICKS | TBS\_VERT | TBS\_BOTH | WS\_BORDER,5,44,17,68,WS\_EX\_TRANSPARENT |
| CTEXT "Vol",IDC\_STATIC\_3,5,113,17,10,WS\_BORDER,WS\_EX\_TRANSPARENT |
| CTEXT "[...]",IDC\_FILE\_NAME,59,6,291,15,SS\_WORDELLIPSIS | WS\_BORDER,WS\_EX\_TRANSPARENT |
| CONTROL "",IDC\_PITCH,"msctls\_trackbar32",TBS\_AUTOTICKS | TBS\_VERT | TBS\_BOTH | WS\_BORDER,25,44,17,68,WS\_EX\_TRANSPARENT |
| CTEXT "Pit",IDC\_STATIC\_4,25,113,17,10,WS\_BORDER,WS\_EX\_TRANSPARENT |
| COMBOBOX IDC\_PLAYLIST,45,46,190,75,CBS\_DROPDOWNLIST | CBS\_DISABLENOSCROLL,WS\_EX\_TRANSPARENT | WS\_EX\_CLIENTEDGE |
| CTEXT "[...]",IDC\_INFO,255,70,115,36,SS\_WORDELLIPSIS | WS\_BORDER,WS\_EX\_TRANSPARENT |
| CTEXT "Info",IDC\_STATIC\_5,255,108,64,10,SS\_WORDELLIPSIS,WS\_EX\_TRANSPARENT |
| PUSHBUTTON "[X]",ID\_X,236,45,20,16,BS\_CENTER | BS\_VCENTER | NOT WS\_TABSTOP,WS\_EX\_CLIENTEDGE | WS\_EX\_STATICEDGE |
| PUSHBUTTON "[N]",ID\_PB\_STATE,236,61,20,16,BS\_CENTER | BS\_VCENTER | NOT WS\_TABSTOP,WS\_EX\_CLIENTEDGE | WS\_EX\_STATICEDGE |
| PUSHBUTTON "[-]",ID\_HIDE\_WINDOW,352,6,20,18,BS\_CENTER | BS\_VCENTER | NOT WS\_TABSTOP,WS\_EX\_CLIENTEDGE | WS\_EX\_STATICEDGE |
| END |
|  |
|  |
| ///////////////////////////////////////////////////////////////////////////// |
| // |
| // DESIGNINFO |
| // |
|  |
| #ifdef APSTUDIO\_INVOKED |
| GUIDELINES DESIGNINFO |
| BEGIN |
| IDD\_MAIN\_DIALOG, DIALOG |
| BEGIN |
| RIGHTMARGIN, 374 |
| END |
| END |
| #endif // APSTUDIO\_INVOKED |

37

|  |
| --- |
| ///////////////////////////////////////////////////////////////////////////// |
| // |
| // AFX\_DIALOG\_LAYOUT |
| // |
|  |
| IDD\_DIALOG1 AFX\_DIALOG\_LAYOUT |
| BEGIN |
| 0 |
| END |
|  |
| IDD\_MAIN\_DIALOG AFX\_DIALOG\_LAYOUT |
| BEGIN |
| 0 |
| END |
|  |
| #endif // Русский (Россия) resources |
| ///////////////////////////////////////////////////////////////////////////// |
|  |
|  |
|  |
| #ifndef APSTUDIO\_INVOKED |
| ///////////////////////////////////////////////////////////////////////////// |
| // |
| // Generated from the TEXTINCLUDE 3 resource. |
| // |
|  |
|  |
| ///////////////////////////////////////////////////////////////////////////// |
| #endif // not APSTUDIO\_INVOKED |

*Лістинг коду з файлу* “audio\_engine.h”

|  |
| --- |
| #pragma once |
| #ifndef \_AUDIO\_ENGINE\_H\_ |
| #define \_AUDIO\_ENGINE\_H\_ |
|  |
| #include "framework.h" |
|  |
| #define \_DEFAULT\_SAMPLE\_RATE\_ 44100 |
| #define \_DEFAULT\_BITS\_PER\_SAMPLE\_ 16 |
|  |
| #define \_CHANNEL\_MONO\_ 0x1 |
| #define \_CHANNEL\_STEREO\_ 0x2 |
|  |
| #define \_NULL\_ID\_ -0x1 |
|  |
| class AudioPlayer; |
|  |
| class AudioEngine { |
| friend class AudioPlayer; |
| public: |
| AudioEngine( |
| AUDIO\_HANDLER fSoundSample = nullptr, |
| AUDIO\_HANDLER fSoundFilter = nullptr |
| ); ~AudioEngine(void); |
|  |
| AudioEngine(const AudioEngine& ae) = delete; |
| AudioEngine& operator=(const AudioEngine& ae) = delete; |

38

|  |
| --- |
| virtual AUDIOID LoadAudioSample(const wchar\_t\* wcWavFile); |
| virtual void PlayAudioSample(AUDIOID ID); |
|  |
|  |
|  |
| virtual bool CreateAudio( |
| DWORD dwSampleRate = \_DEFAULT\_SAMPLE\_RATE\_, |
| WORD wBitsPerSample = \_DEFAULT\_BITS\_PER\_SAMPLE\_, |
| WORD wChannels = \_CHANNEL\_STEREO\_, |
| DWORD dwBlocks = 0x8, DWORD dwBlockSamples = 512 |
| ); |
| virtual bool DestroyAudio(void); |
|  |
| float GetGlobalTime(void) const; |
|  |
| // This class holds loaded sound sample in memory |
| class AudioSample; |
|  |
| // This class represents a sound that is currently playing. It only |
| // holds the sound ID and where this instance of it is up to for its |
| // current playback |
| class PlayingAudio; |
|  |
| private: |
| void waveOutProc(HWAVEOUT hWaveOut, |
| UINT uMsg, DWORD dwParam1, DWORD dwParam2); |
|  |
| static void CALLBACK waveOutProcWrap(HWAVEOUT hWaveOut, |
| UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2 |
| ); |
|  |
| void AudioThread(void); |
|  |
| protected: |
| DWORD m\_dwSampleRate = 0x0; |
| WORD m\_wBitsPerSample = 0x0; |
| WORD m\_wChannels = 0x0; |
|  |
| DWORD m\_dwBlockCount = 0x0, |
| m\_dwBlockSamples = 0x0, |
| m\_dwBlockCurrent = 0x0; |
|  |
| short\* m\_pBlockMemory = nullptr; |
| WAVEHDR\* m\_pWaveHeaders = nullptr; |
| HWAVEOUT m\_hwDevice = nullptr; |
|  |
| std::atomic<float> m\_fGlobalTime = 0.f; |
|  |
| std::thread m\_AudioThread; |
| std::atomic<bool> |
| m\_bAudioThreadActive = false; |
| std::atomic<DWORD> m\_dwBlockFree = 0x0; |
|  |
| std::mutex m\_muxProcessAudio; |
|  |
| std::mutex m\_muxAudioThreadDestroy; |
| std::condition\_variable m\_cvAudioThreadDestroy; |
|  |
| std::mutex m\_muxBlockNotZero; |
| std::condition\_variable m\_cvBlockNotZero; |

39

|  |
| --- |
| AUDIO\_HANDLER |
| m\_fUserSoundSample = nullptr, |
| m\_fUserSoundFilter = nullptr; |
|  |
| // This vector holds all loaded sound samples in memory |
| std::vector<std::shared\_ptr<AudioSample>> |
| vecAudioSamples; |
|  |
| // This list holds all sound that is currently playing |
| std::list<std::shared\_ptr<PlayingAudio>> |
| listActiveSamples; |
|  |
| virtual float GetMixerOutput(int nChannel, |
| float fGlobalTime, float fTimeStep); |
|  |
| template<class AudioSampleType, typename ...ArgumentTypes> |
| AUDIOID LoadAudioSample(const wchar\_t\* wcWavFile, ArgumentTypes&& ...args) { |
| std::shared\_ptr<AudioSampleType> a = std::make\_shared |
| <AudioSampleType>(wcWavFile, std::forward<ArgumentTypes>(args)...); |
|  |
| if (a->m\_bValid) { |
| std::lock\_guard<std::mutex> |
| lgProcessAudio(m\_muxProcessAudio); |
| vecAudioSamples.push\_back( |
| std::move((std::shared\_ptr<AudioSample>)a) |
| ); |
| return(vecAudioSamples.size()); |
| } |
| else |
| { return(\_NULL\_ID\_); } |
| } |
| template<class CreatingPlayingAudioType, typename ...ArgumentTypes> |
| void CreatePlayingAudio(AUDIOID ID, ArgumentTypes&& ...args) { |
| std::shared\_ptr<CreatingPlayingAudioType> s = std::make\_shared |
| <CreatingPlayingAudioType>(ID, std::forward<ArgumentTypes>(args)...); |
|  |
| std::lock\_guard<std::mutex> |
| lgProcessAudio(m\_muxProcessAudio); |
| listActiveSamples.push\_back(std::move(s)); |
| } |
| }; |
|  |
| #endif //\_AUDIO\_ENGINE\_H\_ |

*Лістинг коду з файлу* “audio\_engine.cpp”

|  |
| --- |
| #include "audio\_engine.h" |
|  |
| #include "audio\_sample.h" |
| #include "playing\_audio.h" |
|  |
| #include "logger.h" |
|  |
| AudioEngine::AudioEngine( |
| AUDIO\_HANDLER fSoundSample, |
| AUDIO\_HANDLER fSoundFilter |
| ) : |

40

|  |
| --- |
| m\_fUserSoundSample(fSoundSample), |
| m\_fUserSoundFilter(fSoundFilter) |
| { |
| (void)Logger::LogMessage |
| (Logger::LogLevel::LOG\_LVL\_INFO, |
| L"[Function] : " \_\_FUNCTION\_\_ L" - Success!\t[File] : " \_\_FILE\_\_); |
| } |
| AudioEngine::~AudioEngine(void) { |
| (void)Logger::LogMessage |
| (Logger::LogLevel::LOG\_LVL\_INFO, |
| L"[Function] : " \_\_FUNCTION\_\_ L" - Success!\t[File] : " \_\_FILE\_\_); |
| } |
|  |
| // Load a 16-bit WAVE file @ 44100Hz ONLY into memory. A sample ID |
| // number is returned if successful, otherwise -1 |
| AUDIOID AudioEngine::LoadAudioSample(const wchar\_t\* wcWavFile) { |
| AUDIOID idResult = -(0x1); |
| idResult = LoadAudioSample<AudioSample>(wcWavFile); |
| return(idResult); |
| } |
|  |
| // Add sample 'id' to the mixers sounds to play list |
| void AudioEngine::PlayAudioSample(AUDIOID ID) |
| { CreatePlayingAudio<PlayingAudio>(ID); } |
|  |
| // The audio system uses by default a specific wave format |
| bool AudioEngine::CreateAudio( |
| DWORD dwSampleRate, WORD wBitsPerSample, |
| WORD wChannels, DWORD dwBlocks, |
| DWORD dwBlockSamples |
| ) |
| { |
| // Initialise Sound Engine |
| m\_dwSampleRate = dwSampleRate; |
| m\_wBitsPerSample = wBitsPerSample; |
| m\_wChannels = wChannels; |
|  |
| m\_dwBlockCount = dwBlocks; |
| m\_dwBlockSamples = dwBlockSamples; |
| m\_dwBlockFree = m\_dwBlockCount; |
| m\_dwBlockCurrent = 0x0; |
|  |
| m\_pBlockMemory = nullptr; |
| m\_pWaveHeaders = nullptr; |
|  |
| // Device is available |
| WAVEFORMATEX waveFormat; |
| waveFormat.wFormatTag = WAVE\_FORMAT\_PCM; |
| waveFormat.nSamplesPerSec = m\_dwSampleRate; |
| waveFormat.wBitsPerSample = m\_wBitsPerSample; |
| waveFormat.nChannels = m\_wChannels; |
| waveFormat.nBlockAlign = (waveFormat.wBitsPerSample / 0x8) \* waveFormat.nChannels; |
| waveFormat.nAvgBytesPerSec = waveFormat.nSamplesPerSec \* waveFormat.nBlockAlign; |
| waveFormat.cbSize = 0x0; |

41

|  |
| --- |
| // Open Device if valid |
| if (waveOutOpen(&m\_hwDevice, WAVE\_MAPPER, |
| &waveFormat, (DWORD\_PTR)waveOutProcWrap, |
| (DWORD\_PTR)this, CALLBACK\_FUNCTION) != S\_OK |
| ) |
| { return(DestroyAudio()); } |
|  |
| // Allocate Wave|Block Memory |
| m\_pBlockMemory = new short[m\_dwBlockCount \* m\_dwBlockSamples]; |
| if (m\_pBlockMemory == nullptr) |
| { return(DestroyAudio()); } |
| ZeroMemory(m\_pBlockMemory, (m\_wBitsPerSample / 0x8) \* m\_dwBlockCount \* m\_dwBlockSamples); |
|  |
| m\_pWaveHeaders = new WAVEHDR[m\_dwBlockCount]; |
| if (m\_pWaveHeaders == nullptr) |
| { return(DestroyAudio()); } |
| ZeroMemory(m\_pWaveHeaders, sizeof(WAVEHDR) \* m\_dwBlockCount); |
|  |
| // Link headers to block memory |
| for (size\_t n = 0x0; n < m\_dwBlockCount; n++) { |
| m\_pWaveHeaders[n].dwBufferLength = m\_dwBlockSamples \* (m\_wBitsPerSample / 0x8); |
| m\_pWaveHeaders[n].lpData = (LPSTR)(m\_pBlockMemory + (n \* m\_dwBlockSamples)); |
| } |
|  |
| m\_bAudioThreadActive = true; |
| m\_AudioThread = std::thread(&AudioEngine::AudioThread, this); |
|  |
| return(true); |
| } |
|  |
| // Stop and clean up audio system |
| bool AudioEngine::DestroyAudio(void) { |
| m\_bAudioThreadActive = false; |
|  |
| std::unique\_lock<std::mutex> |
| lm(m\_muxAudioThreadDestroy); |
| m\_cvAudioThreadDestroy.wait(lm); |
|  |
| m\_AudioThread.join(); |
|  |
| return(false); |
| } |

42

|  |
| --- |
| // Handler for soundcard request for more data |
| void AudioEngine::waveOutProc(HWAVEOUT hWaveOut, UINT uMsg, DWORD dwParam1, DWORD dwParam2) { |
| switch (uMsg) |
| { |
| case WOM\_OPEN: |
| { /\*Code...\*/ } break; |
| case WOM\_CLOSE: |
| { /\*Code...\*/ } break; |
|  |
| case WOM\_DONE: { |
| m\_dwBlockFree += 0x1; |
| std::unique\_lock<std::mutex> lm(m\_muxBlockNotZero); |
| m\_cvBlockNotZero.notify\_one(); |
| } break; |
|  |
| default: |
| break; |
| } |
| } |
|  |
| // Static wrapper for sound card handler |
| void CALLBACK AudioEngine::waveOutProcWrap(HWAVEOUT hWaveOut, |
| UINT uMsg, DWORD dwInstance, DWORD dwParam1, DWORD dwParam2 |
| ) |
| { |
| ((AudioEngine\*)dwInstance)-> |
| waveOutProc(hWaveOut, uMsg, dwParam1, dwParam2); |
| } |
|  |
| // Audio thread. This loop responds to requests from the soundcard to fill 'blocks' |
| // with audio data. If no requests are available it goes dormant until the sound |
| // card is ready for more data. The block is fille by the "user" in some manner |
| // and then issued to the soundcard. |
| void AudioEngine::AudioThread(void) { |
| m\_fGlobalTime = 0.f; |
| float fTimeStep = 1.f / (float)m\_dwSampleRate; |
|  |
| // Goofy hack to get maximum integer for a type at run-time |
| short nMaxSample = (short)pow(0x2, m\_wBitsPerSample - 0x1) - 0x1; |
| float fMaxSample = (float)nMaxSample; |
| short nPreviousSample = 0x0; |
|  |
| while (m\_bAudioThreadActive) { |
| // Wait for block to become available |
| if (m\_dwBlockFree == 0x0) { |
| std::unique\_lock<std::mutex> lm(m\_muxBlockNotZero); |
| while (m\_dwBlockFree == 0x0) |
| { m\_cvBlockNotZero.wait(lm); } |
| } |
|  |
| // Block is here, so use it |
| m\_dwBlockFree -= 0x1; |
|  |
| // Prepare block for processing |
| if (m\_pWaveHeaders[m\_dwBlockCurrent].dwFlags & WHDR\_PREPARED) |
| { waveOutUnprepareHeader(m\_hwDevice, &m\_pWaveHeaders[m\_dwBlockCurrent], sizeof(WAVEHDR)); } |

43

|  |
| --- |
| short nNewSample = 0x0; |
| int nCurrentBlock = m\_dwBlockCurrent \* m\_dwBlockSamples; |
|  |
| auto clip = [](float fSample, float fMax) { |
| if (fSample >= 0.f) |
| { return(fmin(fSample, fMax)); } |
| else |
| { return(fmax(fSample, -fMax)); } |
| }; |
|  |
| for (size\_t n = 0x0; n < m\_dwBlockSamples; n += m\_wChannels) { |
| // User Process |
| for (size\_t c = 0x0; c < m\_wChannels; c++) { |
| nNewSample = (short)(clip(GetMixerOutput(c, m\_fGlobalTime, fTimeStep), 1.f) \* fMaxSample); |
| m\_pBlockMemory[nCurrentBlock + n + c] = nNewSample; |
| nPreviousSample = nNewSample; |
| } |
|  |
| m\_fGlobalTime = m\_fGlobalTime + fTimeStep; |
| } |
|  |
| // Send block to sound device |
| waveOutPrepareHeader(m\_hwDevice, &m\_pWaveHeaders[m\_dwBlockCurrent], sizeof(WAVEHDR)); |
| waveOutWrite(m\_hwDevice, &m\_pWaveHeaders[m\_dwBlockCurrent], sizeof(WAVEHDR)); |
|  |
| m\_dwBlockCurrent += 0x1; |
| m\_dwBlockCurrent %= m\_dwBlockCount; |
| } |
|  |
| (void)waveOutClose(m\_hwDevice); |
| m\_fUserSoundSample = m\_fUserSoundFilter = nullptr; |
|  |
| std::unique\_lock<std::mutex> |
| lm(m\_muxAudioThreadDestroy); |
| m\_cvAudioThreadDestroy.notify\_one(); |
| } |

44

|  |
| --- |
| // The Sound Mixer - If the user wants to play many sounds simultaneously, and |
| // perhaps the same sound overlapping itself, then you need a mixer, which |
| // takes input from all sound sources for that audio frame. This mixer maintains |
| // a list of sound locations for all concurrently playing audio samples. Instead |
| // of duplicating audio data, we simply store the fact that a sound sample is in |
| // use and an offset into its sample data. As time progresses we update this offset |
| // until it is beyound the length of the sound sample it is attached to. At this |
| // point we remove the playing souind from the list. |
| float AudioEngine::GetMixerOutput(int nChannel, float fGlobalTime, float fTimeStep) { |
| // Accumulate sample for this channel |
| float fMixerSample = 0.f; |
|  |
| auto fProcessAudio = [&](std::list<std::shared\_ptr<PlayingAudio>>& list) { |
| std::lock\_guard<std::mutex> |
| lgProcessAudio(m\_muxProcessAudio); |
|  |
| for (auto& s : list) { |
| fMixerSample = s->ProcessAudioSample( |
| nChannel, fGlobalTime, fTimeStep, fMixerSample, |
| vecAudioSamples[s->AudioSampleID() - 0x1]); |
| } |
|  |
| // If sounds have completed then remove them |
| list.remove\_if([](std::shared\_ptr<PlayingAudio>& s) |
| { return(s.get()->m\_bFinish.load()); } |
| ); |
| }; fProcessAudio(listActiveSamples); |
|  |
| // The users application might be generating sound, so grab that if it exists |
| if (m\_fUserSoundSample != nullptr) |
| { fMixerSample += m\_fUserSoundSample(nChannel, fGlobalTime, fTimeStep); } |
|  |
| // Return the sample via an optional user override to filter the sound |
| if (m\_fUserSoundFilter != nullptr) |
| { return(m\_fUserSoundFilter(nChannel, fGlobalTime, fMixerSample)); } |
| else { return(fMixerSample); } |
| } |
|  |
| float AudioEngine::GetGlobalTime(void) const |
| { return(m\_fGlobalTime.load()); } |

*Лістинг коду з файлу* “audio\_sample.h”

|  |
| --- |
| #pragma once |
| #ifndef \_AUDIO\_SAMPLE\_H\_ |
| #define \_AUDIO\_SAMPLE\_H\_ |
|  |
| #include "framework.h" |
| #include "audio\_engine.h" |
|  |
| class AudioPlayer; |
|  |
| class AudioEngine::AudioSample |
| { |
| friend class AudioEngine; |
| friend class AudioPlayer; |
|  |
| bool m\_bValid = false; |

45

|  |
| --- |
| public: |
| AudioSample(void); ~AudioSample(void); |
| AudioSample(const wchar\_t\* |
| wcWavFile); |
|  |
| WAVEFORMATEX wavHeader = { 0x0 }; |
|  |
| float\* m\_fSample = nullptr; |
| long m\_nSamples = 0x0; |
| int m\_nChannels = 0x0; |
|  |
| std::wstring m\_wsWavFile; |
|  |
| protected: |
| virtual bool LoadAudioSample |
| (const wchar\_t\* wcWavFile); |
| }; |
|  |
| #endif //\_AUDIO\_SAMPLE\_H\_ |

*Лістинг коду з файлу* “audio\_sample.cpp”

|  |
| --- |
| #include "audio\_sample.h" |
|  |
| #include "logger.h" |
|  |
| AudioEngine::AudioSample::AudioSample(void) |
| { /\*Code...\*/ } |
|  |
| AudioEngine::AudioSample::AudioSample |
| (const wchar\_t\* wcWavFile) |
| { |
| if (wcWavFile == NULL) |
| { m\_bValid = false; |
| { return; } |
| } |
| m\_wsWavFile.append(wcWavFile); |
| m\_bValid = LoadAudioSample |
| (m\_wsWavFile.data()); |
| } |
| AudioEngine::AudioSample::~AudioSample(void) { |
| if (m\_fSample) |
| { delete[] m\_fSample, |
| m\_fSample = nullptr; |
| } |
| } |
|  |
| bool AudioEngine::AudioSample::LoadAudioSample(const wchar\_t\* wcWavFile) { |
| // Get a path to Wav file |
| WCHAR wcFullPath[MAX\_PATH]{}; |
| GetFullPathName(wcWavFile, |
| MAX\_PATH, wcFullPath, NULL); |
|  |
| // Load Wav file and convert to float format |
| FILE\* f = nullptr; |
| \_wfopen\_s(&f, wcFullPath, L"rb"); |
| if (f == nullptr) { return(false); } |

46

|  |
| --- |
| char dump[0x4] = { 0x0 }; |
| std::fread(&dump, sizeof(char), 0x4, f); // Read "RIFF" |
| if (strncmp(dump, "RIFF", 0x4) != 0x0) { return(false); } |
| std::fread(&dump, sizeof(char), 0x4, f); // Not Interested |
| std::fread(&dump, sizeof(char), 0x4, f); // Read "WAVE" |
| if (strncmp(dump, "WAVE", 0x4) != 0x0) { return(false); } |
|  |
| // Read Wave description chunk |
| std::fread(&dump, sizeof(char), 0x4, f); // Read "FMT" |
| std::fread(&dump, sizeof(char), 0x4, f); // Not Interested |
| // Read Wave Format Structure chunk |
| // Note the -2, because the structure has 2 bytes to indicate its own size |
| // which are not in the wav file |
| std::fread(&wavHeader, sizeof(WAVEFORMATEX) - 0x2, 0x1, f); |
|  |
| // Just check if wave format is compatible with AE |
| if (wavHeader.wBitsPerSample != 0x10 || wavHeader.nSamplesPerSec != 44100) |
| { std::fclose(f); return(false); } |
|  |
| // Search for audio data chunk |
| long nChunksize = 0x0; |
| std::fread(&dump, sizeof(char), 0x4, f); // Read chunk header |
| std::fread(&nChunksize, sizeof(long), 0x1, f); // Read chunk size |
| while (strncmp(dump, "data", 0x4) != 0x0) { |
| // Not audio data, so just skip it |
| std::fseek(f, nChunksize, SEEK\_CUR); |
| std::fread(&dump, sizeof(char), 0x4, f); |
| std::fread(&nChunksize, sizeof(long), 0x1, f); |
| } |
|  |
| // Finally got to data, so read it all in and convert to float samples |
| m\_nSamples = nChunksize / (wavHeader.nChannels \* (wavHeader.wBitsPerSample >> 0x3)); |
| m\_nChannels = wavHeader.nChannels; |
|  |
| // Create floating point buffer to hold audio sample |
| m\_fSample = new float[m\_nSamples \* m\_nChannels]; |
| float\* pSample = m\_fSample; |
|  |
| // Read in audio data and normalise |
| for (long i = 0x0; i < m\_nSamples; i++) |
| for (int c = 0x0; c < m\_nChannels; c++) { |
| short s = 0x0; |
| std::fread(&s, sizeof(short), 0x1, f); |
| \*pSample = (float)s / (float)(MAXSHORT); |
| pSample += 0x1; |
| } |
|  |
| // All done, flag sound as valid |
| std::fclose(f); |
|  |
| return(true); |
| } |

47

*Лістинг коду з файлу* “playing\_audio.h”

|  |
| --- |
| #pragma once |
| #ifndef \_PlAYING\_AUDIO\_SAMPLE\_H\_ |
| #define \_PlAYING\_AUDIO\_SAMPLE\_H\_ |
|  |
| #include "audio\_engine.h" |
|  |
| class AudioPlayer; |
|  |
| class AudioEngine::AudioSample; |
|  |
| class AudioEngine::PlayingAudio |
| { |
| friend class AudioEngine; |
| friend class AudioPlayer; |
|  |
| AUDIOID m\_nAudioSampleID = 0x0; |
| public: |
| PlayingAudio |
| (AUDIOID nAudioSampleID = -(0x1)); |
| ~PlayingAudio(void); |
|  |
| std::atomic<double> |
| m\_dSamplePosition = 0.0; |
| std::atomic<bool> |
| m\_bFinish = false; |
|  |
| AUDIOID AudioSampleID(void) const; |
|  |
| protected: |
| virtual float ProcessAudioSample(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const std::shared\_ptr<AudioEngine::AudioSample>& pS |
| ); |
| }; |
|  |
| #endif //\_PlAYING\_AUDIO\_SAMPLE\_H\_ |

*Лістинг коду з файлу* “playing\_audio.cpp”

|  |
| --- |
| #include "playing\_audio.h" |
|  |
| #include "audio\_sample.h" |
|  |
| AudioEngine::PlayingAudio::PlayingAudio(AUDIOID nAudioSampleID) : |
| m\_nAudioSampleID(nAudioSampleID) |
| { |
| m\_dSamplePosition = 0.0; |
| m\_bFinish = false; |
| } |
| AudioEngine::PlayingAudio::~PlayingAudio(void) |
| { /\*Code...\*/ } |
|  |
| AUDIOID AudioEngine::PlayingAudio::AudioSampleID(void) const |
| { return(m\_nAudioSampleID); } |

48

|  |
| --- |
| float AudioEngine::PlayingAudio::ProcessAudioSample(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const std::shared\_ptr<AudioEngine::AudioSample>& pS |
| ) |
| { |
| // Calculate sample position |
| m\_dSamplePosition.store( |
| m\_dSamplePosition.load() + (double)pS->wavHeader.nSamplesPerSec \* fTimeStep |
| ); |
|  |
| // If sample position is valid add to the mix |
| if (m\_dSamplePosition < pS.get()->m\_nSamples) |
| { fMixerSample += pS.get()->m\_fSample[((long)round(m\_dSamplePosition) \* |
| pS.get()->m\_nChannels) + nChannel]; |
| } |
| else |
| { m\_bFinish = true; } // Else sound has completed |
| return(fMixerSample); |
| } |

*Лістинг коду з файлу* “audio\_player.h”

|  |
| --- |
| #pragma once |
| #ifndef \_AUDIO\_PLAYER\_H\_ |
| #define \_AUDIO\_PLAYER\_H\_ |
|  |
| #include "audio\_engine.h" |
|  |
| #include "playing\_audio.h" |
|  |
| class AudioEngine::AudioSample; |
|  |
| class AudioEngine::PlayingAudio; |
|  |
| class AudioPlayer { |
|  |
| protected: |
| typedef AudioEngine\* |
| pAudioEngine; |
|  |
| typedef AudioEngine::AudioSample\* |
| pAudioSample; |
| typedef AudioEngine::PlayingAudio\* |
| pPlayingAudio; |
| private: |
| class AudioPlayerHandler : |
| public AudioEngine::PlayingAudio |
| { |
| friend class AudioPlayer; |
|  |
| typedef AudioPlayer\* |
| pAudioPlayer; |
| public: |
| AudioPlayerHandler(AUDIOID nAudioSampleID, |
| pAudioPlayer pAP |
| ); |
| ~AudioPlayerHandler(void); |

49

|  |
| --- |
| float ProcessAudioSample(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const std::shared\_ptr<AudioEngine::AudioSample>& pS |
| ); |
| protected: |
| pAudioPlayer m\_pAP = nullptr; |
| }; |
|  |
| public: |
| AudioPlayer |
| (pAudioEngine pAE); |
| ~AudioPlayer(void); |
|  |
| AudioPlayer(const AudioPlayer& ae) = delete; |
| AudioPlayer& operator=(const AudioPlayer& ae) = delete; |
|  |
| struct AE\_DATA { |
| DWORD dwSampleRate = 0x0; |
| WORD wBitsPerSample = 0x0, |
| wChannels = 0x0; |
|  |
| DWORD dwBlockCount = 0x0, |
| dwBlockSamples = 0x0; |
| }; |
|  |
| typedef std::pair<std::shared\_ptr |
| <AudioEngine::AudioSample>, std::shared\_ptr<AudioEngine::PlayingAudio> |
| > AUDIO\_DATA; |
|  |
| protected: |
| pAudioEngine m\_pAE = nullptr; |
|  |
| AUDIO\_DATA CreateAudio |
| (AUDIOID nAudioSampleID); |
|  |
| virtual float AudioHandler(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const pAudioSample pS, const pPlayingAudio pH |
| ) = 0x0; |
|  |
| void AudioEngineData |
| (AE\_DATA& ae\_data) const; |
| }; |
|  |
| #endif // \_AUDIO\_PLAYER\_H\_ |

*Лістинг коду з файлу* “audio\_player.cpp”

|  |
| --- |
| #include "audio\_player.h" |
|  |
| #include "audio\_sample.h" |
|  |
| AudioPlayer::AudioPlayer(pAudioEngine pAE) : |
| m\_pAE(pAE) |
| { |
| /\*Code...\*/ |
| } |

50

|  |
| --- |
| AudioPlayer::~AudioPlayer(void) |
| { /\*Code...\*/ } |
|  |
| AudioPlayer::AUDIO\_DATA AudioPlayer::CreateAudio |
| (AUDIOID nAudioSampleID) |
| { |
| m\_pAE->CreatePlayingAudio |
| <AudioPlayerHandler>(nAudioSampleID, this); |
| return( |
| AUDIO\_DATA( |
| m\_pAE->vecAudioSamples[nAudioSampleID - 0x1], |
| m\_pAE->listActiveSamples.back() |
| ) |
| ); |
| } |
|  |
|  |
|  |
| AudioPlayer::AudioPlayerHandler::AudioPlayerHandler(AUDIOID nAudioSampleID, |
| pAudioPlayer pAP) : PlayingAudio(nAudioSampleID), m\_pAP(pAP) |
| { |
| m\_dSamplePosition = 0.0; |
| m\_bFinish = false; |
| } |
| AudioPlayer::AudioPlayerHandler::~AudioPlayerHandler(void) { /\*Code...\*/ } |
|  |
| float AudioPlayer::AudioPlayerHandler::ProcessAudioSample(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const std::shared\_ptr<AudioEngine::AudioSample>& pS |
| ) |
| { |
| float fResult = 0.f; |
| if (m\_bFinish) { goto linkExit; } |
| fResult = m\_pAP->AudioHandler |
| (nChannel, fGlobalTime, fTimeStep, |
| fMixerSample, pS.get(), this |
| ); |
| linkExit: |
| return(fResult); |
| } |
|  |
| void AudioPlayer::AudioEngineData |
| (AE\_DATA& ae\_data) const |
| { |
| AE\_DATA \_ae\_data\_ = { |
| m\_pAE->m\_dwSampleRate, |
| m\_pAE->m\_wBitsPerSample, |
| m\_pAE->m\_wChannels, |
|  |
| m\_pAE->m\_dwBlockCount, |
| m\_pAE->m\_dwBlockSamples |
| }; ae\_data = \_ae\_data\_; |
| } |

51

*Лістинг коду з файлу* “main\_audio\_player.h”

|  |
| --- |
| #pragma once |
| #ifndef \_MAIN\_AUDIO\_PLAYER\_H\_ |
| #define \_MAIN\_AUDIO\_PLAYER\_H\_ |
|  |
| #include "audio\_player.h" |
|  |
| class MainAudioPlayer : |
| public AudioPlayer |
| { |
|  |
| public: |
| MainAudioPlayer |
| (pAudioEngine pAE); |
| ~MainAudioPlayer(void); |
|  |
| MainAudioPlayer(const MainAudioPlayer& ae) = delete; |
| MainAudioPlayer& operator=(const MainAudioPlayer& ae) = delete; |
|  |
| const wchar\_t\* |
| FileName(void) const; |
|  |
| enum { |
| STATE\_PLAY = 0x1, |
| STATE\_STOP = 0x0, |
| STATE\_NULL = -0x1 |
| }; |
|  |
| AUDIOID LoadAudioSample |
| (const wchar\_t\* wcWavFile); |
| bool ChangeCurrentAudioSample |
| (AUDIOID ID); |
| AUDIOID CurrentAudioSample(void) const; |
|  |
| void ChangeStateAudio |
| (signed int nState); |
| void SwapStateAudio(void); |
| signed int CurrentStateAudio |
| (void) const; |
|  |
| void VolumeAudio |
| (float fVolume); |
| float CurrentVolume(void) const; |
|  |
| void PitchAudio |
| (float fPitch); |
| float CurrentPitch(void) const; |
|  |
| void PositonAudio |
| (double dSamplePosition); |
| double CurrentPositonAudio |
| (void) const; |
|  |
| DWORD NumOfSamples |
| (void) const; |
| DWORD NumOfSamplesPerSec |
| (void) const; |

52

|  |
| --- |
| private: |
| mutable std::vector |
| <AUDIO\_DATA> m\_vecAudio; |
| mutable std::atomic<AUDIOID> m\_nCurrentAudio = \_NULL\_ID\_; |
|  |
| std::atomic |
| <signed int> m\_nState = STATE\_NULL; |
| std::atomic<float> |
| m\_fVolume = 1.f, |
| m\_fPitch = 1.f; |
|  |
| mutable std::recursive\_mutex m\_muxData; |
|  |
| virtual float AudioHandler(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const pAudioSample pS, const pPlayingAudio pH |
| ); |
|  |
| AUDIO\_DATA& |
| AudioData(void) const; |
| std::shared\_ptr<AudioEngine::AudioSample>& |
| AudioSample(void) const; |
| std::shared\_ptr<AudioEngine::PlayingAudio>& |
| PlayingAudio(void) const; |
|  |
| float Clip |
| (float& f); |
| double Clip |
| (double& d); |
| }; |
|  |
| #endif // \_MAIN\_AUDIO\_PLAYER\_H\_ |

*Лістинг коду з файлу* “main\_audio\_player.cpp”

|  |
| --- |
| #include "main\_audio\_player.h" |
|  |
| #include "audio\_engine.h" |
| #include "audio\_sample.h" |
| #include "playing\_audio.h" |
|  |
| MainAudioPlayer::MainAudioPlayer(pAudioEngine pAE) : |
| AudioPlayer(pAE) |
| { |
| /\*Code...\*/ |
| } |
| MainAudioPlayer::~MainAudioPlayer(void) |
| { |
| /\*Code...\*/ |
| } |
|  |
| AudioPlayer::AUDIO\_DATA& |
| MainAudioPlayer::AudioData(void) const |
| { |
| return(m\_vecAudio |
| [m\_nCurrentAudio.load() - 0x1] |
| ); |
| } |

53

|  |
| --- |
| std::shared\_ptr<AudioEngine::AudioSample>& |
| MainAudioPlayer::AudioSample(void) const |
| { |
| return(m\_vecAudio |
| [m\_nCurrentAudio.load() - 0x1].first |
| ); |
| } |
| std::shared\_ptr<AudioEngine::PlayingAudio>& |
| MainAudioPlayer::PlayingAudio(void) const |
| { |
| return(m\_vecAudio |
| [m\_nCurrentAudio.load() - 0x1].second |
| ); |
| } |
|  |
| AUDIOID MainAudioPlayer::LoadAudioSample |
| (const wchar\_t\* wcWavFile) |
| { |
| AUDIOID ID = \_NULL\_ID\_; |
| ID = m\_pAE-> |
| LoadAudioSample(wcWavFile); |
|  |
| if (ID == \_NULL\_ID\_) |
| { return(ID); } |
|  |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| AUDIO\_DATA ad = CreateAudio(ID); |
|  |
| m\_vecAudio.push\_back(ad); |
| m\_nCurrentAudio.store |
| (m\_vecAudio.size()); |
|  |
| m\_nState.store(STATE\_STOP); |
| return(m\_nCurrentAudio.load()); |
| } |
|  |
| bool MainAudioPlayer::ChangeCurrentAudioSample |
| (AUDIOID ID) |
| { |
| bool bResult = 0x0; |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| if (bResult = (ID <= (AUDIOID)m\_vecAudio.size())) |
| { m\_nCurrentAudio.store(ID); } |
| return(bResult); |
| } |
|  |
| AUDIOID MainAudioPlayer::CurrentAudioSample(void) const |
| { return(m\_nCurrentAudio.load()); } |
|  |
| const wchar\_t\* MainAudioPlayer::FileName(void) const { |
| if (m\_nCurrentAudio.load() == \_NULL\_ID\_) { return(NULL); } |
|  |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |

54

|  |
| --- |
| return(AudioSample().get()-> |
| m\_wsWavFile.data()); |
| } |
|  |
| void MainAudioPlayer::SwapStateAudio(void) |
| { if (m\_nState.load() != STATE\_NULL) |
| { m\_nState = !m\_nState; } |
| } |
| void MainAudioPlayer::ChangeStateAudio(signed int nState) |
| { m\_nState.store(nState); } |
| signed int MainAudioPlayer::CurrentStateAudio(void) const |
| { return(m\_nState.load()); } |
|  |
|  |
|  |
| void MainAudioPlayer::VolumeAudio(float fVolume) { |
| fVolume = Clip(fVolume); |
| m\_fVolume.store(fVolume); |
| } |
| float MainAudioPlayer::CurrentVolume(void) const |
| { return(m\_fVolume.load()); } |
|  |
| void MainAudioPlayer::PitchAudio(float fPitch) { |
| fPitch = fmax(fPitch, 0.f); |
| m\_fPitch.store(fPitch); |
| } |
| float MainAudioPlayer::CurrentPitch(void) const |
| { return(m\_fPitch.load()); } |
|  |
|  |
|  |
| void MainAudioPlayer::PositonAudio(double dSamplePosition) { |
| if (m\_nCurrentAudio.load() == \_NULL\_ID\_) { return; } |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| dSamplePosition = Clip(dSamplePosition); |
| PlayingAudio().get()->m\_dSamplePosition = (double)AudioSample().get()-> |
| m\_nSamples \* dSamplePosition; |
| } |
|  |
| double MainAudioPlayer::CurrentPositonAudio(void) const { |
| if (m\_nCurrentAudio.load() == \_NULL\_ID\_) { return(0.0); } |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| return(PlayingAudio().get()->m\_dSamplePosition / |
| AudioSample().get()->m\_nSamples |
| ); |
| } |
|  |
| DWORD MainAudioPlayer::NumOfSamples(void) const { |
| if (m\_nCurrentAudio.load() == \_NULL\_ID\_) { return(0x0); } |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| return(AudioSample().get()-> |
| m\_nSamples |
| ); |
| } |

55

|  |
| --- |
| DWORD MainAudioPlayer::NumOfSamplesPerSec(void) const { |
| if (m\_nCurrentAudio.load() == \_NULL\_ID\_) { return(0x0); } |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| return(AudioSample().get()-> |
| wavHeader.nSamplesPerSec |
| ); |
| } |
|  |
|  |
|  |
| float MainAudioPlayer::AudioHandler(int nChannel, |
| float fGlobalTime, float fTimeStep, float fMixerSample, |
| const pAudioSample pS, const pPlayingAudio pH |
| ) |
| { |
| std::lock\_guard<std::recursive\_mutex> |
| lgData(m\_muxData); |
|  |
| if (m\_nCurrentAudio.load() == \_NULL\_ID\_ || |
| PlayingAudio().get() != pH |
| ) |
| { return(fMixerSample); } |
|  |
| AE\_DATA ae\_data = { 0x0 }; |
| AudioEngineData(ae\_data); |
| if (pS->m\_nChannels <= nChannel) |
| { return(fMixerSample); } |
| int nDelta = pS->m\_nChannels - ae\_data.wChannels; |
| for (int i = 0x0; i <= max(nDelta, 0x0); i++) { |
| // Calculate sample position |
| if (m\_nState.load() != STATE\_STOP) { |
| PlayingAudio().get()->m\_dSamplePosition.store(PlayingAudio().get()->m\_dSamplePosition.load() + |
| (double)(pS->wavHeader.nSamplesPerSec / pS->m\_nChannels) \* m\_fPitch.load() \* fTimeStep |
| ); |
| } else |
| { return(fMixerSample); } |
|  |
| // If sample position is valid add to the mix |
| if (PlayingAudio().get()->m\_dSamplePosition < pS->m\_nSamples) { |
| fMixerSample += ( |
| pS->m\_fSample[(long)(PlayingAudio().get()->m\_dSamplePosition) \* pS->m\_nChannels + nChannel] |
| ) \* m\_fVolume.load(); |
| } |
| else { |
| PlayingAudio().get()-> |
| m\_dSamplePosition = (double)pS->m\_nSamples; |
| } // Else sound has completed |
| } |
| return(fMixerSample); |
| } |

56

|  |
| --- |
| float MainAudioPlayer::Clip(float& f) { |
| if (f >= 0.f) |
| { f = fmin(f, 1.f); } |
| else |
| { f = fmax(f, 0.f); } |
| return(f); |
| } |
|  |
| double MainAudioPlayer::Clip(double& d) { |
| if (d >= 0.f) |
| { d = fmin(d, 1.0); } |
| else |
| { d = fmax(d, 0.0); } |
| return(d); |
| } |

*Лістинг коду з файлу* “logger.h”

|  |
| --- |
| #pragma once |
| #ifndef \_LOGGER\_H\_ |
| #define \_LOGGER\_H\_ |
|  |
| #include "framework.h" |
|  |
| #define HPRINT(\_data\_) std::wcout << \_data\_ << std::endl; |
|  |
| class Logger { |
|  |
| public: |
| Logger(void); ~Logger(void); |
|  |
| static Logger& GetInstance(void); |
|  |
| Logger(const Logger&) = delete; |
| Logger& operator=(const Logger&) = delete; |
|  |
| static void LoadLogLevel(DWORD dwLogLevel); |
|  |
| static BOOL LogMessage(DWORD dwLogLevel, |
| const std::wstring& wsMessage); |
| static DWORD LogLastError(DWORD dwLogLevel); |
| static DWORD LogFormatError(DWORD dwLogLevel, |
| DWORD dwError); |
|  |
| static DWORD ClearLog(void); |
|  |
|  |
|  |
| static DWORD ShowLastError(const wchar\_t\* wcTitle); |
| static DWORD ShowFormatError(const wchar\_t\* wcTitle, |
| DWORD dwError); |
| static void ShowMessage(const wchar\_t\* wcMessage, |
| const wchar\_t\* wcTitle); |
|  |
| static void LogAndShowMessage(DWORD dwLogLevel, |
| const std::wstring& wsMessage); |

57

|  |
| --- |
| enum LogLevel { |
| LOG\_LVL\_NULL = 0x0, |
| LOG\_LVL\_INFO = 0x1, |
| LOG\_LVL\_WARNING = 0x2, |
| LOG\_LVL\_ERROR = 0x3, |
| LOG\_LVL\_DEBUG = 0x4 |
| }; |
|  |
| private: |
| HWND m\_hwnd = NULL; |
| DWORD m\_dwLogLevel = LogLevel::LOG\_LVL\_NULL; |
|  |
| std::mutex m\_mLogMessage; |
| }; |
|  |
| #endif //\_LOGGER\_H\_ |

*Лістинг коду з файлу* “logger.cpp”

|  |
| --- |
| #include "logger.h" |
|  |
| Logger::Logger(void) |
| { /\*Code...\*/ } |
| Logger::~Logger(void) |
| { /\*Code...\*/ } |
|  |
| Logger& Logger::GetInstance(void) |
| { static Logger \_h; return(\_h); } |
|  |
| void Logger::LoadLogLevel(DWORD dwLogLevel) { |
| std::lock\_guard<std::mutex> |
| lgLogMessage(GetInstance().m\_mLogMessage); |
|  |
| GetInstance().m\_dwLogLevel = dwLogLevel; |
| } |
|  |
| BOOL Logger::LogMessage(DWORD dwLogLevel, const std::wstring& wsMessage) { |
| if (dwLogLevel == LogLevel::LOG\_LVL\_NULL) |
| { return(FALSE); } |
| std::lock\_guard<std::mutex> |
| lgLogMessage(GetInstance().m\_mLogMessage); |
|  |
| size\_t sizeMessageLength = wsMessage.size(); |
|  |
| SYSTEMTIME sysTime = { 0x0 }; |
|  |
| HANDLE hLogFileHandle = INVALID\_HANDLE\_VALUE; |
| DWORD dwEndOfFile = 0x0, dwNumberOfBytesWritten = 0x0; |
|  |
| DWORD dwError = ERROR\_SUCCESS; |
|  |
| if (GetInstance().m\_dwLogLevel < dwLogLevel) { return(FALSE); } |
| if (sizeMessageLength <= NULL || |
| sizeMessageLength >= 1024) { return(FALSE); } |
|  |
| //\*wstring wsDateTime, wsSeverityTag, wsFormattedMessage; |

58

|  |
| --- |
| wchar\_t wcDateTime[64]{}; |
| wchar\_t wcSeverityTag[16]{}; |
| wchar\_t wcFormattedMessage[1024]; |
|  |
| GetLocalTime(&sysTime); |
|  |
| swprintf\_s(wcDateTime, L"[%i/%i/%i %i:%i:%i.%i]", |
| sysTime.wMonth, sysTime.wDay, sysTime.wYear, |
| sysTime.wHour, sysTime.wMinute, sysTime.wSecond, |
| sysTime.wMilliseconds); |
|  |
| switch (dwLogLevel) { |
| case LogLevel::LOG\_LVL\_INFO: |
| wcscpy\_s(wcSeverityTag, L"[INFO] : "); |
| break; |
| case LogLevel::LOG\_LVL\_WARNING: |
| wcscpy\_s(wcSeverityTag, L"[WARNING] : "); |
| break; |
| case LogLevel::LOG\_LVL\_ERROR: |
| wcscpy\_s(wcSeverityTag, L"[ERROR] : "); |
| break; |
| case LogLevel::LOG\_LVL\_DEBUG: |
| wcscpy\_s(wcSeverityTag, L"[DEBUG] : "); |
| break; |
|  |
| default: |
| break; |
| } |
|  |
| swprintf\_s(wcFormattedMessage, L"%s\n", wsMessage.data()); |
|  |
| if ((hLogFileHandle = CreateFile(LOG\_FILE\_NAME, FILE\_APPEND\_DATA, |
| FILE\_SHARE\_READ, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL)) == INVALID\_HANDLE\_VALUE) |
| { dwError = GetLastError(); goto linkError; } |
|  |
| dwEndOfFile = SetFilePointer(hLogFileHandle, 0x0, NULL, FILE\_END); |
|  |
| if(!WriteFile(hLogFileHandle, wcDateTime, wcslen(wcDateTime) \* sizeof(wchar\_t), |
| &dwNumberOfBytesWritten, NULL)) |
| { dwError = GetLastError(); goto linkError; } |
|  |
| if(!WriteFile(hLogFileHandle, wcSeverityTag, wcslen(wcSeverityTag) \* sizeof(wchar\_t), |
| &dwNumberOfBytesWritten, NULL)) |
| { dwError = GetLastError(); goto linkError; } |
|  |
| if(!WriteFile(hLogFileHandle, wcFormattedMessage, wcslen(wcFormattedMessage) \* sizeof(wchar\_t), |
| &dwNumberOfBytesWritten, NULL)) |
| { dwError = GetLastError(); goto linkError; } |
|  |
| if (hLogFileHandle != INVALID\_HANDLE\_VALUE) |
| if(!CloseHandle(hLogFileHandle)) |
| { dwError = GetLastError(); goto linkExit; } |

59

|  |
| --- |
| linkExit: |
| return(TRUE); |
| linkError: |
| (void)ShowFormatError |
| (L"LogMessage - Error!", dwError); { return(FALSE); } |
| } |
|  |
| DWORD Logger::LogLastError(DWORD dwLogLevel) { |
| DWORD dwError = ERROR\_SUCCESS; |
| (void)LogFormatError(dwLogLevel, |
| dwError = GetLastError()); |
| return(dwError); |
| } |
|  |
| DWORD Logger::LogFormatError(DWORD dwLogLevel, DWORD dwError) { |
| WCHAR wcError[256]{}; |
| FormatMessage(FORMAT\_MESSAGE\_FROM\_SYSTEM, NULL, dwError, |
| MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), wcError, 256, NULL); |
| (void)LogMessage(dwLogLevel, |
| L"[" + std::to\_wstring(dwError) + L"]" + wcError); |
| return(dwError); |
| } |
|  |
| DWORD Logger::ClearLog(void) { |
| DWORD dwError = ERROR\_SUCCESS; |
|  |
| if(!DeleteFile(LOG\_FILE\_NAME)) |
| { dwError = GetLastError(); } |
|  |
| return(dwError); |
| } |
|  |
|  |
|  |
| DWORD Logger::ShowLastError(const wchar\_t\* wcTitle) { |
| DWORD dwError = ERROR\_SUCCESS; |
| ShowFormatError(wcTitle, dwError = GetLastError()); |
| return(dwError); |
| } |
| DWORD Logger::ShowFormatError(const wchar\_t\* wcTitle, DWORD dwError) { |
| WCHAR wcError[256]{}; |
| FormatMessage(FORMAT\_MESSAGE\_FROM\_SYSTEM, NULL, dwError, |
| MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT), wcError, 256, NULL); |
|  |
| wchar\_t wcMessage[256]{}; |
| swprintf\_s(wcMessage, L"[%d]%s", dwError, wcError); |
|  |
| ShowMessage(wcMessage, wcTitle); |
| return(dwError); |
| } |
| void Logger::ShowMessage(const wchar\_t\* wcMessage, const wchar\_t\* wcTitle) { |
| (void)MessageBeep(MB\_ICONERROR), (void)MessageBox(NULL, wcMessage, wcTitle, |
| MB\_OK | MB\_DEFBUTTON1 | MB\_APPLMODAL | MB\_SETFOREGROUND); |
| } |

60

|  |
| --- |
| void Logger::LogAndShowMessage(DWORD dwLogLevel, const std::wstring& wsMessage) { |
| if(LogMessage(dwLogLevel, wsMessage)) |
| { ShowMessage(wsMessage.data(), |
| L"Log-Message"); |
| } |
| } |

*Лістинг коду з файлу* “basedlgbox.h”

|  |
| --- |
| #ifndef \_BASEDLGBOX\_H\_ |
| #define \_BASEDLGBOX\_H\_ |
|  |
| #include "framework.h" |
|  |
| class BaseDlgBox { |
| LPWSTR m\_dlgResName = NULL; |
| HWND m\_hParent = NULL; |
|  |
| public: |
| BaseDlgBox(LPWSTR dlgResName); |
| virtual ~BaseDlgBox(void); |
|  |
| HWND GetDialogHWND(void) const; |
|  |
| virtual BOOL CreateDlg |
| (HINSTANCE hInstance, HWND hParent); |
| virtual BOOL DestroyDlg |
| (int nExitCode); |
| virtual BOOL ShowDlg(int nCmdShow); |
|  |
| virtual bool |
| OnUserCreate(void) = 0x0; |
| virtual bool |
| OnUserDestroy(void) = 0x0; |
|  |
| private: |
| static LRESULT CALLBACK WindowProc(HWND \_In\_ hWnd, UINT \_In\_ uMsg, |
| WPARAM \_In\_ wParam, LPARAM \_In\_ lParam); |
| protected: |
| HINSTANCE m\_hInstance = NULL; |
|  |
| HWND m\_hWnd = NULL; |
|  |
| virtual LRESULT CALLBACK HandleMessage(UINT \_In\_ uMsg, |
| WPARAM \_In\_ wParam, LPARAM \_In\_ lParam); |
| }; |
|  |
| #endif //\_BASEDLGBOX\_H\_ |

61

*Лістинг коду з файлу* “basedlgbox.cpp”

|  |
| --- |
| #include "basedlgbox.h" |
|  |
| #include "logger.h" |
|  |
| BaseDlgBox::BaseDlgBox(LPWSTR dlgResName) : |
| m\_dlgResName(dlgResName) { /\*Code...\*/ } |
| BaseDlgBox::~BaseDlgBox(void) { |
| //Code... |
| } |
| HWND BaseDlgBox::GetDialogHWND(void) const { return(m\_hWnd); } |
| BOOL BaseDlgBox::CreateDlg |
| (HINSTANCE hInstance, HWND hParent) |
| { |
| BOOL bResult = 0x1; |
|  |
| if(!m\_hInstance && !m\_hParent) |
| { m\_hInstance = hInstance; |
| m\_hParent = hParent; |
| } |
| else { bResult = 0x0; goto linkExit; } |
|  |
| m\_hWnd = CreateDialog(m\_hInstance, m\_dlgResName, m\_hParent, |
| reinterpret\_cast<DLGPROC>(WindowProc)); |
| if (!m\_hWnd) { bResult = 0x0; goto linkExit; } |
|  |
| SetWindowLongPtr(m\_hWnd, GWLP\_USERDATA, |
| reinterpret\_cast<LONG\_PTR>(this)); |
|  |
| if(OnUserCreate()) { (void)Logger::LogMessage |
| (Logger::LogLevel::LOG\_LVL\_INFO, |
| L"[Function] : OnUserCreate" L" - Success!\t[File] : " \_\_FILE\_\_); |
| } |
| else { (void)Logger::LogMessage |
| (Logger::LogLevel::LOG\_LVL\_INFO, |
| L"[Function] : OnUserCreate" L" - Failed!\t[File] : " \_\_FILE\_\_); |
| } |
|  |
| linkExit: |
| return(bResult); |
| } |
| BOOL BaseDlgBox::DestroyDlg(int nExitCode) { |
| BOOL bResult = 0x1; |
| if (!m\_hWnd) |
| { bResult = 0x0; |
| goto linkExit; |
| } |
|  |
| bResult = EndDialog(m\_hWnd, nExitCode); |
| SetWindowLongPtr(m\_hWnd, GWLP\_USERDATA, |
| reinterpret\_cast<LONG\_PTR>(nullptr)); |
| m\_hWnd = NULL; |
|  |
| if(OnUserDestroy()) { (void)Logger::LogMessage |
| (Logger::LogLevel::LOG\_LVL\_INFO, |
| L"[Function] : OnUserDestroy" L" - Success!\t[File] : " \_\_FILE\_\_); |
| } |

62

|  |
| --- |
| else { (void)Logger::LogMessage |
| (Logger::LogLevel::LOG\_LVL\_INFO, |
| L"[Function] : OnUserDestroy" L" - Failed!\t[File] : " \_\_FILE\_\_); |
| } |
|  |
| linkExit: |
| return(bResult); |
| } |
| BOOL BaseDlgBox::ShowDlg(int nCmdShow) { |
| BOOL bResult = ShowWindow |
| (m\_hWnd, nCmdShow); |
| return(bResult); |
| } |
| LRESULT CALLBACK BaseDlgBox::WindowProc(\_In\_ HWND hWnd, \_In\_ UINT uMsg, \_In\_ WPARAM wParam, \_In\_ LPARAM lParam) { |
| BaseDlgBox\* pBaseDlgBox = nullptr; |
|  |
| pBaseDlgBox = reinterpret\_cast<BaseDlgBox\*> |
| (GetWindowLongPtr(hWnd, GWLP\_USERDATA)); |
| if (uMsg == WM\_INITDIALOG) { return(0x1); } |
|  |
| if (pBaseDlgBox) |
| { return(pBaseDlgBox->HandleMessage(uMsg, wParam, lParam)); } |
| else { return(0x0); } |
| } |
| LRESULT CALLBACK BaseDlgBox::HandleMessage(UINT \_In\_ uMsg, |
| WPARAM \_In\_ wParam, LPARAM \_In\_ lParam) { |
| switch (uMsg) { |
|  |
| case WM\_CLOSE: |
| EndDialog(m\_hWnd, wParam); m\_hWnd = NULL; |
| return(TRUE); |
|  |
| default: |
| break; |
| } |
| return(0x0); |
| } |

*Лістинг коду з файлу* “maindlg.h”

|  |
| --- |
| #ifndef \_MAINDLG\_H\_ |
| #define \_MAINDLG\_H\_ |
|  |
| #include "framework.h" |
| #include "basedlgbox.h" |
|  |
| class AudioEngine; |
|  |
| class Button; |
| class StaticText; |
|  |
| class Combobox; |
| class Slider; |
|  |
| #include "main\_audio\_player.h" |

class MainDlg :

public BaseDlgBox

{

63

|  |
| --- |
| AudioEngine& m\_refAE; |
|  |
| MainAudioPlayer m\_audioPlayer; |
|  |
| wchar\_t\* m\_wcWavFile = NULL; |
|  |
| bool NewAudioMusic |
| (const wchar\_t\* wcWavFile); |
| bool ChangeAudioMusic( |
| AUDIOID nMusicID |
| ); |
| public: |
| MainDlg(LPWSTR dlgResName, |
| const wchar\_t\* wcWavFile, AudioEngine& refAE); |
| ~MainDlg(void); |
| private: |
| LRESULT CALLBACK HandleMessage(UINT \_In\_ uMsg, |
| WPARAM \_In\_ wParam, LPARAM \_In\_ lParam); |
|  |
| bool OnUserCreate(void); bool OnUserDestroy(void); |
|  |
| enum { |
| PB\_STATE\_STOP = 0x0, |
| PB\_STATE\_REPEAT = 0x1, |
| PB\_STATE\_CONTINUE = 0x2 |
| }; |
| INT m\_iPB = 0x0; |
|  |
| std::unique\_ptr |
| <Button> m\_pPlay = nullptr; |
| std::unique\_ptr |
| <Button> m\_pPBState = nullptr; |
|  |
| std::unique\_ptr |
| <Combobox> m\_pPlayList = nullptr; |
|  |
| struct { |
| std::unique\_ptr<StaticText> |
| pInfo = nullptr, |
| pFileName = nullptr, |
| pDuration = nullptr; |
| } m\_conStaticText; |
|  |
| struct { |
| std::unique\_ptr |
| <Slider> pAudioTrack = nullptr; |
| std::unique\_ptr |
| <Slider> pVolume = nullptr; |
| std::unique\_ptr |
| <Slider> pPitch = nullptr; |
| } m\_conSlider; |
|  |
| bool m\_bHold = 0x0; |
|  |
| bool WavFileName |
| (wchar\_t wcFileName[MAX\_PATH]); |

64

|  |
| --- |
| void AudioDuration(wchar\_t wcAudioDuration[MAX\_PATH], |
| double dCurrentPositionAudio, float fNumOfSamples, float fNumOfSamplesPerSec); |
| }; |
|  |
| #endif //\_MAINDLG\_H\_ |

*Лістинг коду з файлу* “maindlg.cpp”

|  |
| --- |
| #include "maindlg.h" |
| #include "resource.h" |
|  |
| #include "control\_button.h" |
| #include "control\_combobox.h" |
|  |
| #include "control\_static\_text.h" |
| #include "control\_slider.h" |
|  |
| #include "audio\_engine.h" |
|  |
| #include "logger.h" |
|  |
| #define \_TIMER\_MAIN\_ID\_ (INT)0x0 |
| #define \_TIMER\_MAIN\_ELAPSE\_ (INT)128 |
|  |
| #define \_FILE\_FORMAT\_ L"\*.wav\n" |
| #define \_NULL\_STRING\_ L"[NULL\_STATE]" |
|  |
| #define \_NULL\_DURATION\_ L"00:00:00" |
| #define \_DURATION\_ L"%02i:%02i:%02i" |
|  |
| MainDlg::MainDlg(LPWSTR dlgResName, const wchar\_t\* wcWavFile, AudioEngine& refAE) : |
| m\_wcWavFile((wchar\_t\*)wcWavFile), m\_refAE(refAE), m\_audioPlayer(&m\_refAE), BaseDlgBox(dlgResName) |
| { |
| /\*Code...\*/ |
| } |
| MainDlg::~MainDlg(void) { /\*Code...\*/ } |
|  |
| bool MainDlg::OnUserCreate(void) { |
| DragAcceptFiles |
| (m\_hWnd, TRUE); |
| m\_pPBState = std::make\_unique |
| <Button>(m\_hWnd, ID\_PB\_STATE, |
| L"[/]", 0x1 |
| ); |
|  |
| m\_pPlay = std::make\_unique |
| <Button>(m\_hWnd, ID\_PLAY, |
| \_NULL\_STRING\_, 0x0 |
| ); |
|  |
| m\_pPlayList = std::make\_unique |
| <Combobox>(m\_hWnd, IDC\_PLAYLIST, 0x0 |
| ); |
|  |
| m\_conStaticText.pInfo = std::make\_unique |
| <StaticText>(m\_hWnd, IDC\_INFO, |
| \_NULL\_STRING\_, 0x0 |
| ); |

65

|  |
| --- |
| m\_conStaticText.pFileName = std::make\_unique |
| <StaticText>(m\_hWnd, IDC\_FILE\_NAME, |
| \_NULL\_STRING\_, 0x0 |
| ); |
| m\_conStaticText.pDuration = std::make\_unique |
| <StaticText>(m\_hWnd, IDC\_DURATION, |
| \_NULL\_DURATION\_, 0x0 |
| ); |
|  |
| m\_conSlider.pAudioTrack = std::make\_unique |
| <Slider>(m\_hWnd, IDC\_AUDIO\_TRACK, 0x0, POINT{ 0x0, 1000 }); |
| m\_conSlider.pVolume = std::make\_unique |
| <Slider>(m\_hWnd, IDC\_VOLUME, 50, POINT{ 0x0, 100 }); |
| m\_conSlider.pPitch = std::make\_unique |
| <Slider>(m\_hWnd, IDC\_PITCH, 0x2, POINT{ 0x1, 0x3 }); |
|  |
| (void)NewAudioMusic(m\_wcWavFile); |
|  |
| (void)SetTimer(m\_hWnd, |
| \_TIMER\_MAIN\_ID\_, \_TIMER\_MAIN\_ELAPSE\_, NULL |
| ); |
|  |
| return(true); |
| } |
| bool MainDlg::OnUserDestroy(void) { return(true); } |
|  |
| bool MainDlg::NewAudioMusic(const wchar\_t\* wcWavFile) { |
| if (wcWavFile == NULL) { return(false); } |
|  |
| AUDIOID nMusicID = m\_audioPlayer. |
| LoadAudioSample(wcWavFile); |
|  |
| if (nMusicID == -(0x1)) |
| { Logger::ShowMessage(wcWavFile, |
| L"Failed to Load File!"); return(false); |
| } |
|  |
| m\_audioPlayer.ChangeStateAudio |
| (MainAudioPlayer::STATE\_STOP); |
| (void)ChangeAudioMusic(nMusicID); |
|  |
| WCHAR wcFileName[MAX\_PATH]; |
| WCHAR wcExt[MAX\_PATH]; |
|  |
| \_wsplitpath\_s(wcWavFile, NULL, 0x0, NULL, 0x0, wcFileName, MAX\_PATH, wcExt, MAX\_PATH); |
|  |
| m\_pPlayList.get()-> |
| AddItemString(wcFileName); |
| m\_pPlayList.get()->ChangeState(0x1); |
|  |
| return(true); |
| } |
|  |
| bool MainDlg::ChangeAudioMusic( |
| AUDIOID nMusicID |
| ) |

66

|  |
| --- |
| { |
| bool bResult = false; |
|  |
| if (nMusicID != \_NULL\_ID\_) |
| { bResult = true; } |
| (void)m\_audioPlayer. |
| ChangeCurrentAudioSample(nMusicID); |
|  |
| m\_audioPlayer.PositonAudio(NULL); |
|  |
| m\_conStaticText.pFileName.get()-> |
| ChangeState(nMusicID != \_NULL\_ID\_); |
| m\_conStaticText.pFileName.get()-> |
| SetText(nMusicID != \_NULL\_ID\_ ? m\_audioPlayer.FileName() : \_NULL\_STRING\_); |
|  |
| m\_pPlayList.get()-> |
| SelectItem(nMusicID != \_NULL\_ID\_ ? |
| nMusicID - 0x1 : -0x1 |
| ); |
|  |
| m\_conStaticText.pInfo.get()-> |
| ChangeState(nMusicID != \_NULL\_ID\_); |
| std::wstring wsInfo = L"Info..."; |
| m\_conStaticText.pInfo.get()->SetText(nMusicID != \_NULL\_ID\_ ? wsInfo.data() : \_NULL\_STRING\_); |
|  |
| m\_pPlay.get()-> |
| ChangeState(nMusicID != \_NULL\_ID\_); |
| WCHAR\* wcState[0x2] = { (wchar\_t\*)L"Play", (wchar\_t\*)L"Pause" }; |
| m\_pPlay.get()->SetText |
| (nMusicID != \_NULL\_ID\_ ? |
| wcState[m\_audioPlayer.CurrentStateAudio()] : |
| \_NULL\_STRING\_ |
| ); |
|  |
| m\_conStaticText.pDuration.get()-> |
| ChangeState(nMusicID != \_NULL\_ID\_); |
| m\_conStaticText.pDuration.get()->SetText(\_NULL\_DURATION\_); |
|  |
| m\_audioPlayer.VolumeAudio( |
| (float)(m\_conSlider.pVolume.get()->GetRange() - m\_conSlider.pVolume.get()->GetPos()) / |
| m\_conSlider.pVolume.get()->GetRange() |
| ); |
| m\_audioPlayer.PitchAudio( |
| ((m\_conSlider.pPitch.get()->GetRangeParam().y - m\_conSlider.pPitch.get()->GetPos() + |
| m\_conSlider.pPitch.get()->GetRangeParam().x) \* 0.5f) |
| ); |
|  |
| return(bResult); |
| } |
|  |
| LRESULT CALLBACK MainDlg::HandleMessage(UINT \_In\_ uMsg, |
| WPARAM \_In\_ wParam, LPARAM \_In\_ lParam) |
| { |

67

|  |
| --- |
| switch (uMsg) |
| { |
|  |
| case WM\_COMMAND: { |
| UNREFERENCED\_PARAMETER(lParam); |
| UINT lwID = LOWORD(wParam); |
|  |
| switch (lwID) { |
| case ID\_FILE: { |
| WCHAR wcFileName[MAX\_PATH] = { 0x0 }; |
| BOOL bResult = WavFileName(wcFileName); |
|  |
| if (bResult == TRUE) |
| { (void)NewAudioMusic |
| (wcFileName); |
| } |
| } break; |
| case ID\_PLAY: { |
| if (m\_audioPlayer.CurrentAudioSample() != \_NULL\_ID\_) { |
| switch (m\_audioPlayer.CurrentStateAudio()) { |
| case MainAudioPlayer::STATE\_PLAY: |
| { m\_pPlay.get()->SetText(L"Play"); } break; |
| case MainAudioPlayer::STATE\_STOP: |
| { m\_pPlay.get()->SetText(L"Pause"); } break; |
| } |
| m\_audioPlayer.SwapStateAudio(); |
| } |
| } break; |
| case ID\_PB\_STATE: { |
| m\_iPB = (m\_iPB + 0x1) % 0x3; |
| WCHAR\* wcState[0x3] = { |
| (wchar\_t\*)L"[/]", |
| (wchar\_t\*)L"[∞]", |
| (wchar\_t\*)L"[→]" |
| }; |
| m\_pPBState.get()->SetText(wcState[m\_iPB]); |
| } break; |
| case ID\_X: { |
| m\_audioPlayer.ChangeStateAudio |
| (MainAudioPlayer::STATE\_STOP); |
| (void)ChangeAudioMusic(\_NULL\_ID\_); |
| } break; |
|  |
| case ID\_HIDE\_WINDOW: { |
| BOOL bVisible = IsWindowVisible(m\_hWnd); |
| ShowWindow(m\_hWnd, !bVisible ? |
| SW\_MAXIMIZE : SW\_MINIMIZE |
| ); |
| } break; |
|  |
| default: |
| break; |
| } |
|  |
| UINT hwID = HIWORD(wParam); |
|  |
| switch (hwID) { |

68

|  |
| --- |
| case CBN\_SELCHANGE: { |
| HWND hWndCombobox = (HWND)lParam; |
| if (hWndCombobox == m\_pPlayList.get()->GetHandle()) { |
| INT ID = m\_pPlayList.get()->SelectedItemID() + 0x1; |
| if (m\_audioPlayer.CurrentAudioSample() != ID) { |
| m\_audioPlayer.ChangeStateAudio |
| (MainAudioPlayer::STATE\_STOP); |
| (void)ChangeAudioMusic(ID); |
| } |
| } |
| } break; |
|  |
| default: |
| break; |
| } |
|  |
| } break; |
|  |
| case WM\_HSCROLL: { |
| HWND hWndTrack = (HWND)lParam; |
| if (hWndTrack == m\_conSlider.pAudioTrack.get()->GetHandle()) |
| { |
| if (LOWORD(wParam) != SB\_ENDSCROLL && |
| LOWORD(wParam) != SB\_THUMBPOSITION) |
| { |
| DWORD dwPos = m\_conSlider.pAudioTrack.get()->GetPos(); |
| m\_audioPlayer.PositonAudio(dwPos / |
| (float)m\_conSlider.pAudioTrack.get()->GetRange()); |
| m\_bHold = 0x1; |
|  |
| WCHAR wcDur[MAX\_PATH] = { 0x0 }; |
|  |
| double dCurrentPositionAudio = m\_audioPlayer. |
| CurrentPositonAudio(); |
| float fNumOfSamples = (float)m\_audioPlayer. |
| NumOfSamples(); |
| float fNumOfSamplesPerSec = (float)m\_audioPlayer. |
| NumOfSamplesPerSec(); |
|  |
| AudioDuration(wcDur, |
| dCurrentPositionAudio, fNumOfSamples, fNumOfSamplesPerSec); |
| m\_conStaticText.pDuration.get()->SetText(wcDur); |
| } |
| else { m\_bHold = 0x0; } |
| } |
| } break; |
| case WM\_VSCROLL: { |
| HWND hWndTrack = (HWND)lParam; |
| if (hWndTrack == m\_conSlider.pVolume.get()->GetHandle()) |
| { |
| if (LOWORD(wParam) != SB\_ENDSCROLL) { |
| DWORD dwPos = m\_conSlider.pVolume.get()->GetPos(); |
| m\_audioPlayer.VolumeAudio((float)(m\_conSlider.pVolume.get()->GetRange() - dwPos) / |
| m\_conSlider.pVolume.get()->GetRange()); |
| } |
| } |

69

|  |
| --- |
| if (hWndTrack == m\_conSlider.pPitch.get()->GetHandle()) |
| { |
| if (LOWORD(wParam) != SB\_ENDSCROLL) { |
| DWORD dwPos = m\_conSlider.pPitch.get()->GetPos(); |
| m\_audioPlayer.PitchAudio( |
| ((m\_conSlider.pPitch.get()->GetRangeParam().y - dwPos + |
| m\_conSlider.pPitch.get()->GetRangeParam().x) \* 0.5f) |
| ); |
| } |
| } |
| } break; |
|  |
| case WM\_TIMER: { |
| INT nID = (INT)wParam; |
| switch (nID) |
| { |
|  |
| case \_TIMER\_MAIN\_ID\_: { |
| if (m\_bHold) { break; } |
| if (m\_audioPlayer.CurrentAudioSample() != \_NULL\_ID\_) { |
| if (m\_conSlider.pAudioTrack.get()->GetPos() == |
| m\_conSlider.pAudioTrack.get()->GetRange() |
| ) |
| { |
| m\_audioPlayer.PositonAudio(0.0); |
| switch (m\_iPB) { |
| case PB\_STATE\_STOP: { |
| m\_audioPlayer.ChangeStateAudio |
| (MainAudioPlayer::STATE\_STOP); |
| m\_pPlay.get()->SetText(L"Play"); |
| } break; |
| case PB\_STATE\_REPEAT: { |
| m\_audioPlayer.ChangeStateAudio |
| (MainAudioPlayer::STATE\_PLAY); |
| } break; |
| case PB\_STATE\_CONTINUE: { |
| INT iCount = m\_pPlayList. |
| get()->CountOfItems(); |
| m\_audioPlayer.ChangeStateAudio |
| (MainAudioPlayer::STATE\_PLAY); |
| if (iCount > 0x1) { |
| INT ID = (m\_audioPlayer. |
| CurrentAudioSample() % iCount |
| ); |
| (void)ChangeAudioMusic(ID + 0x1); |
| } |
| } break; |
|  |
| default: |
| break; |
| } |
| } |
| m\_conSlider.pAudioTrack.get()->SetPos |
| ((DWORD)(m\_audioPlayer.CurrentPositonAudio() \* |
| m\_conSlider.pAudioTrack.get()->GetRange()) |
| ); |
|  |
| WCHAR wcDur[MAX\_PATH] = { 0x0 }; |

70

|  |
| --- |
| double dCurrentPositionAudio = m\_audioPlayer. |
| CurrentPositonAudio(); |
| float fNumOfSamples = (float)m\_audioPlayer. |
| NumOfSamples(); |
| float fNumOfSamplesPerSec = (float)m\_audioPlayer. |
| NumOfSamplesPerSec(); |
|  |
| AudioDuration(wcDur, |
| dCurrentPositionAudio, fNumOfSamples, fNumOfSamplesPerSec); |
| m\_conStaticText.pDuration.get()->SetText(wcDur); |
| } |
| } |
|  |
| default: |
| break; |
| } |
| } |
|  |
| case WM\_DROPFILES: { |
| HDROP hDrop = (HDROP)wParam; |
| if (hDrop != NULL) { |
| wchar\_t wcFileName[MAX\_PATH]{}; |
|  |
| size\_t sCnt = DragQueryFile(hDrop, |
| 0xFFFFFFFF, NULL, 0x0 |
| ); |
| for (size\_t i = 0x0; i < sCnt; i++) { |
| DragQueryFile(hDrop, |
| i, wcFileName, MAX\_PATH |
| ); |
| (void)NewAudioMusic(wcFileName); |
| ZeroMemory(&wcFileName, |
| sizeof(wcFileName) |
| ); |
| } |
| DragFinish(hDrop); |
| } |
| } break; |
|  |
| case WM\_PAINT: { |
| UNREFERENCED\_PARAMETER(lParam), UNREFERENCED\_PARAMETER(wParam); |
| PAINTSTRUCT psPaint; |
| HDC hdc = BeginPaint(m\_hWnd, &psPaint); |
| FillRect(hdc, &psPaint.rcPaint, |
| reinterpret\_cast<HBRUSH>(GetStockObject(WHITE\_BRUSH))); |
| (void)EndPaint(m\_hWnd, &psPaint); |
| } break; |
|  |
|  |
|  |
| case WM\_CLOSE: { |
| UNREFERENCED\_PARAMETER(lParam), UNREFERENCED\_PARAMETER(wParam); |
| if(!DestroyDlg(EXIT\_SUCCESS)) |
| { Logger::ShowLastError |
| (L"DestroyDlg(nExitCode)"); |
| } |
| PostQuitMessage(EXIT\_SUCCESS); |
| } break; |

71

|  |
| --- |
| default: |
| break; |
| } |
| return(0x0); |
| } |
|  |
| bool MainDlg::WavFileName |
| (wchar\_t wcFileName[MAX\_PATH]) |
| { |
| BOOL bResult = 0x0; |
|  |
| OPENFILENAMEW hFile; |
| ZeroMemory(&hFile, |
| sizeof(hFile) |
| ); |
| hFile.hwndOwner = m\_hWnd; |
|  |
| hFile.lpstrFile = wcFileName; |
| hFile.nMaxFile = MAX\_PATH; |
|  |
| hFile.lpstrFilter = \_FILE\_FORMAT\_; |
| hFile.lpstrFileTitle = NULL; |
| hFile.nMaxFileTitle = 0x0; |
|  |
| hFile.lpstrInitialDir = NULL; |
|  |
| hFile.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST; |
| hFile.lStructSize = sizeof(hFile); |
|  |
| bResult = GetSaveFileName(&hFile); |
| return(bResult); |
| } |
|  |
| void MainDlg::AudioDuration(wchar\_t wcAudioDuration[MAX\_PATH], |
| double dCurrentPositionAudio, float fNumOfSamples, float fNumOfSamplesPerSec) |
| { |
| size\_t sElapsedSec = (size\_t)round(dCurrentPositionAudio \* |
| (fNumOfSamples / fNumOfSamplesPerSec)); |
| size\_t sElapsedMin = (size\_t) |
| (sElapsedSec / 60.f); |
| size\_t sElapsedHour = (size\_t) |
| (sElapsedMin / 60.f); |
|  |
| ZeroMemory(wcAudioDuration, |
| sizeof(wchar\_t) \* MAX\_PATH); |
| (void)swprintf\_s((wchar\_t\*)wcAudioDuration, MAX\_PATH, \_DURATION\_, |
| sElapsedHour % 60, sElapsedMin % 60, sElapsedSec % 60); |
| } |

72

*Лістинг коду з файлу* “control.h”

|  |
| --- |
| #pragma once |
| #ifndef \_CONTROL\_H\_ |
| #define \_CONTROL\_H\_ |
|  |
| #include "framework.h" |
|  |
| class Control { |
|  |
| public: |
| Control(HWND hParent, |
| DWORD dwDlgItem); |
| Control(HWND hParent, |
| DWORD dwDlgItem, BOOL bState); |
| virtual ~Control(void) = 0x0; |
|  |
| HWND GetHandle(void) const; |
| DWORD GetItemID(void) const; |
|  |
| enum State { |
| STATE\_ENABLE = 0x1, |
| STATE\_DISABLE = 0x0 |
| }; |
| BOOL ChangeState(bool bSate); |
| protected: |
| HWND m\_hWnd = NULL, |
| m\_hParent = NULL; |
| DWORD m\_dwDlgItem = 0x0; |
| }; |
|  |
| #endif //\_CONTROL\_H\_ |

*Лістинг коду з файлу* “control.cpp”

|  |
| --- |
| #include "control.h" |
|  |
| Control::Control(HWND hParent, DWORD dwDlgItem) : |
| m\_hParent(hParent), m\_dwDlgItem(dwDlgItem) |
| { |
| m\_hWnd = GetDlgItem |
| (m\_hParent, m\_dwDlgItem); |
| } |
| Control::Control(HWND hParent, DWORD dwDlgItem, BOOL bState) : |
| m\_hParent(hParent), m\_dwDlgItem(dwDlgItem) |
| { |
| m\_hWnd = GetDlgItem |
| (m\_hParent, m\_dwDlgItem); |
| (void)ChangeState(bState); |
| } |
| Control::~Control(void) { /\*Code...\*/ } |
|  |
| HWND Control::GetHandle(void) const |
| { return(m\_hWnd); } |
| DWORD Control::GetItemID(void) const |
| { return(m\_dwDlgItem); } |
|  |
| BOOL Control::ChangeState(bool bSate) |
| { return(EnableWindow(m\_hWnd, bSate)); } |

*Лістинг коду з файлу* “control\_button.h”

73

|  |
| --- |
| #pragma once |
| #ifndef \_CONTROL\_BUTTON\_H\_ |
| #define \_CONTROL\_BUTTON\_H\_ |
|  |
| #include "control.h" |
|  |
| class Button : |
| public Control |
| { |
| wchar\_t m\_wcText[MAX\_PATH]{}; |
| public: |
| Button(HWND hParent, DWORD dwDlgItem, |
| const wchar\_t\* wcText, bool bState = STATE\_ENABLE); |
| ~Button(void); |
|  |
| const wchar\_t\* GetText(void); |
|  |
| void SetText |
| (const wchar\_t\* wcText); |
| }; |
|  |
| #endif // \_CONTROL\_STATIC\_TEXT\_H\_ |

*Лістинг коду з файлу* “control\_button.cpp”

|  |
| --- |
| #include "control\_button.h" |
|  |
| Button::Button(HWND hParent, DWORD dwDlgItem, |
| const wchar\_t\* wcText, bool bState) : Control(hParent, dwDlgItem, bState) |
| { |
| if (wcText == NULL) |
| { wcText = (wchar\_t\*)L'\0'; } |
| wcscpy\_s(m\_wcText, MAX\_PATH, wcText); |
| (void)SetWindowText(m\_hWnd, m\_wcText); |
| } |
| Button::~Button(void) { /\*Code...\*/ } |
|  |
| const wchar\_t\* Button::GetText(void) |
| { return(m\_wcText); } |
|  |
| void Button::SetText(const wchar\_t\* wcText) { |
| if (wcscmp(m\_wcText, wcText) == NULL) |
| { return; } |
| if (wcslen(m\_wcText) != NULL) |
| { ZeroMemory(m\_wcText, |
| sizeof(wchar\_t) \* MAX\_PATH); |
| } |
|  |
| if (wcText == NULL) |
| { wcText = (wchar\_t\*)L'\0'; } |
|  |
| wcscpy\_s(m\_wcText, MAX\_PATH, wcText); |
| (void)SetWindowText(m\_hWnd, |
| m\_wcText |
| ); |
| } |

74

*Лістинг коду з файлу* “control\_static\_text.h”

|  |
| --- |
| #pragma once |
| #ifndef \_CONTROL\_STATIC\_TEXT\_H\_ |
| #define \_CONTROL\_STATIC\_TEXT\_H\_ |
|  |
| #include "control.h" |
|  |
| class StaticText : |
| public Control |
| { |
| wchar\_t m\_wcText[MAX\_PATH]{}; |
| public: |
| StaticText(HWND hParent, DWORD dwDlgItem, |
| const wchar\_t\* wcText, bool bState = STATE\_ENABLE); |
| ~StaticText(void); |
|  |
| const wchar\_t\* GetText(void); |
|  |
| void SetText |
| (const wchar\_t\* wcText); |
| }; |
|  |
| #endif // \_CONTROL\_STATIC\_TEXT\_H\_ |

*Лістинг коду з файлу* “control\_static\_text.cpp”

|  |
| --- |
| #include "control\_static\_text.h" |
|  |
| StaticText::StaticText(HWND hParent, DWORD dwDlgItem, |
| const wchar\_t\* wcText, bool bState) : Control(hParent, dwDlgItem, bState) |
| { |
| if (wcText == NULL) |
| { wcText = (wchar\_t\*)L'\0'; } |
| wcscpy\_s(m\_wcText, MAX\_PATH, wcText); |
| (void)SetWindowText(m\_hWnd, |
| m\_wcText |
| ); |
| } |
| StaticText::~StaticText(void) { /\*Code...\*/ } |
|  |
| const wchar\_t\* StaticText::GetText(void) |
| { return(m\_wcText); } |
|  |
| void StaticText::SetText(const wchar\_t\* wcText) { |
| if (wcscmp(m\_wcText, wcText) == NULL) |
| { return; } |
| if (wcslen(m\_wcText) != NULL) |
| { ZeroMemory(m\_wcText, |
| sizeof(wchar\_t) \* MAX\_PATH); |
| } |
|  |
| if (wcText == NULL) |
| { wcText = (wchar\_t\*)L'\0'; } |
|  |
| wcscpy\_s(m\_wcText, MAX\_PATH, wcText); |
| (void)SetWindowText(m\_hWnd, |
| m\_wcText |
| ); |
| } |

75

*Лістинг коду з файлу* “control\_combobox.h”

|  |
| --- |
| #pragma once |
| #ifndef \_CONTROL\_COMBOBOX\_H\_ |
| #define \_CONTROL\_COMBOBOX\_H\_ |
|  |
| #include "control.h" |
|  |
| class Combobox : |
| public Control |
| { |
| std::wstring m\_wsText; |
| public: |
| Combobox(HWND hParent, DWORD dwDlgItem, |
| bool bState = STATE\_ENABLE |
| ); |
| ~Combobox(void); |
|  |
| INT CountOfItems(void); |
| void AddItemString |
| (const wchar\_t\* wcString); |
| const wchar\_t\* ItemString(INT ID); |
| INT SelectedItemID(void); |
|  |
| INT SelectItem(INT ID); |
| }; |
|  |
| #endif // \_CONTROL\_COMBOBOX\_H\_ |

*Лістинг коду з файлу* “control\_combobox.cpp”

|  |
| --- |
| #include "control\_combobox.h" |
|  |
| Combobox::Combobox(HWND hParent, DWORD dwDlgItem, |
| bool bState |
| ) : |
| Control(hParent, dwDlgItem, bState) |
| { |
| /\*Code...\*/ |
| } |
| Combobox::~Combobox(void) { /\*Code...\*/ } |
|  |
| INT Combobox::CountOfItems(void) { |
| INT iCount = CB\_ERR; |
| iCount = SendMessage(m\_hWnd, |
| CB\_GETCOUNT, 0x0, 0x0 |
| ); |
| return(iCount); |
| } |
|  |
| void Combobox::AddItemString |
| (const wchar\_t\* wcString) |

76

|  |
| --- |
| { |
| SendMessage(m\_hWnd, |
| CB\_ADDSTRING, NULL, (LPARAM)wcString |
| ); |
| INT iCount = CountOfItems(); |
| SendMessage(m\_hWnd, |
| CB\_SETCURSEL, iCount - 0x1, 0x0 |
| ); |
| } |
| const wchar\_t\* Combobox::ItemString(INT ID) { |
| if (!m\_wsText.empty()) |
| { m\_wsText.clear(); } |
|  |
| if(ID == -0x1) |
| { return(NULL); } |
| size\_t sLength = SendMessage(m\_hWnd, |
| CB\_GETLBTEXTLEN, ID, NULL |
| ); |
| m\_wsText.reserve(sLength); |
| SendMessage(m\_hWnd, |
| CB\_GETLBTEXT, ID, (LPARAM)m\_wsText.data() |
| ); |
| return(m\_wsText.data()); |
| } |
| INT Combobox::SelectedItemID(void) { |
| INT ID = CB\_ERR; |
| ID = (INT)SendMessage(m\_hWnd, |
| CB\_GETCURSEL, NULL, NULL |
| ); |
| return(ID); |
| } |
|  |
| INT Combobox::SelectItem(INT ID) { |
| INT iResult = CB\_ERR; |
| iResult = SendMessage(m\_hWnd, |
| CB\_SETCURSEL, ID, 0x0 |
| ); |
| return(iResult); |
| } |

*Лістинг коду з файлу* “control\_slider.h”

|  |
| --- |
| #pragma once |
| #ifndef \_CONTROL\_SLIDER\_H\_ |
| #define \_CONTROL\_SLIDER\_H\_ |
|  |
| #include "control.h" |
|  |
| class Slider : |
| public Control |
| { |
| DWORD m\_dwPos = 0x0; |
| POINT m\_pRange = { 0x0 }; |
| public: |
| Slider(HWND hParent, |
| DWORD dwDlgItem, DWORD dwPos, POINT pRange); |
| ~Slider(void); |
|  |
| DWORD GetPos(void); |

77

|  |
| --- |
| void SetPos |
| (DWORD dwPos); |
|  |
| DWORD GetRange(void) const; |
| POINT GetRangeParam |
| (void) const; |
| }; |
|  |
| #endif // \_CONTROL\_SLIDER\_H\_ |

*Лістинг коду з файлу* “control\_slider.cpp”

|  |
| --- |
| #include "control\_slider.h" |
|  |
| Slider::Slider(HWND hParent, DWORD dwDlgItem, DWORD dwPos, POINT pRange) : |
| Control(hParent, dwDlgItem), m\_pRange(pRange) |
| { |
| SendMessage(m\_hWnd, TBM\_SETRANGE, |
| (WPARAM)TRUE, (LPARAM)MAKELONG(m\_pRange.x, m\_pRange.y)); |
|  |
| SendMessage(m\_hWnd, TBM\_SETPAGESIZE, |
| 0x0, (LPARAM)0xA); |
|  |
| SendMessage( |
| m\_hWnd, TBM\_SETPOS, |
| (WPARAM)TRUE, (LPARAM)(m\_dwPos = dwPos) |
| ); |
| } |
| Slider::~Slider(void) { /\*Code...\*/ } |
|  |
| DWORD Slider::GetPos(void) { |
| m\_dwPos = SendMessage( |
| m\_hWnd, TBM\_GETPOS, 0x0, 0x0 |
| ); |
| return(m\_dwPos); |
| } |
|  |
| void Slider::SetPos(DWORD dwPos) { |
| SendMessage( |
| m\_hWnd, TBM\_SETPOS, |
| (WPARAM)TRUE, (LPARAM)(m\_dwPos = dwPos) |
| ); |
| } |
|  |
| DWORD Slider::GetRange(void) const |
| { return(m\_pRange.y - m\_pRange.x); } |
|  |
| POINT Slider::GetRangeParam(void) const |
| { return(m\_pRange); } |

78

**ДОДАТОК Б**

****