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

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”*.
2. 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.
3. Програма повинна бути здатною відтворювати аудіофайли, які відповідають даним вимогам:
    1. Частота дискретизації: *44100 Гц*;
    2. Бітрейт: *16 біт*;
    3. Аудиоканал: *Mono/Stereo*;
4. Програма повинна мати 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.;

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 C++ 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.

ЗМІСТ

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.

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:

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.

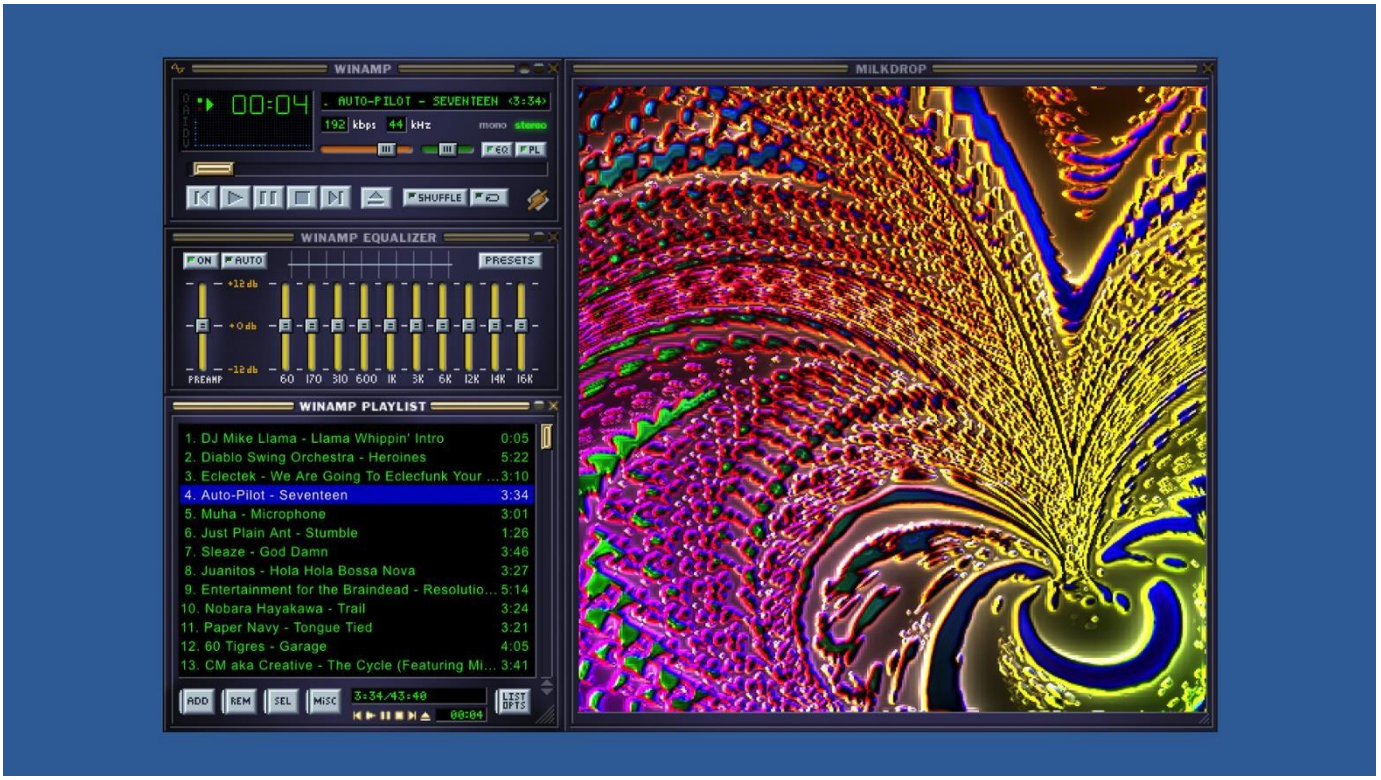


Рис.1.1. Вигляд аудіоплеєру ***Winamp*** у версії для *Windows 10/11*

Вочевидь можна виділити нестандартний дизайн та перелік можливостей, що в теперішніх реаліях цілком відповідає потребам користувача.

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.

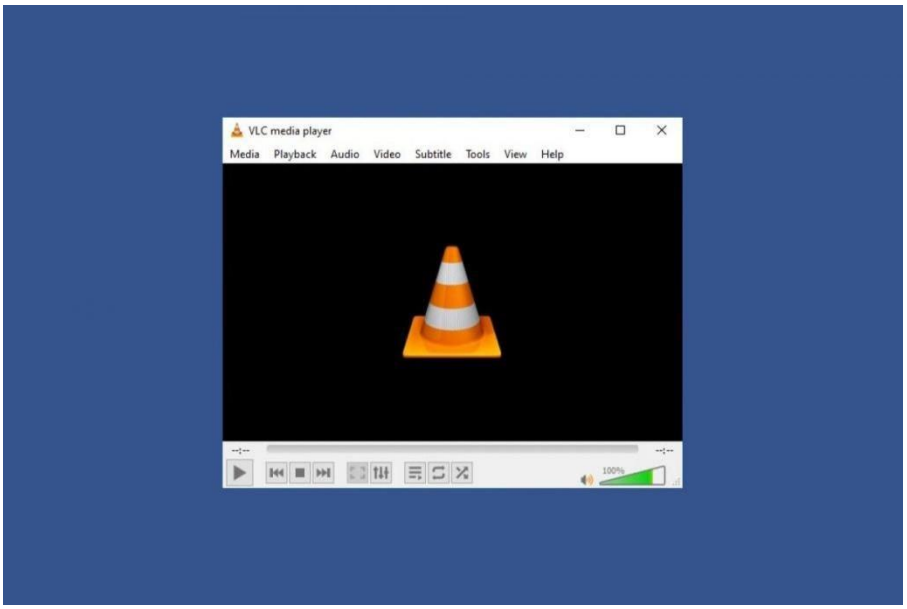
Рис.1.2. Вигляд аудіоплеєру ***AIMP***

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***

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***



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 аудіостримінгового сервісу ***Soundcloud***

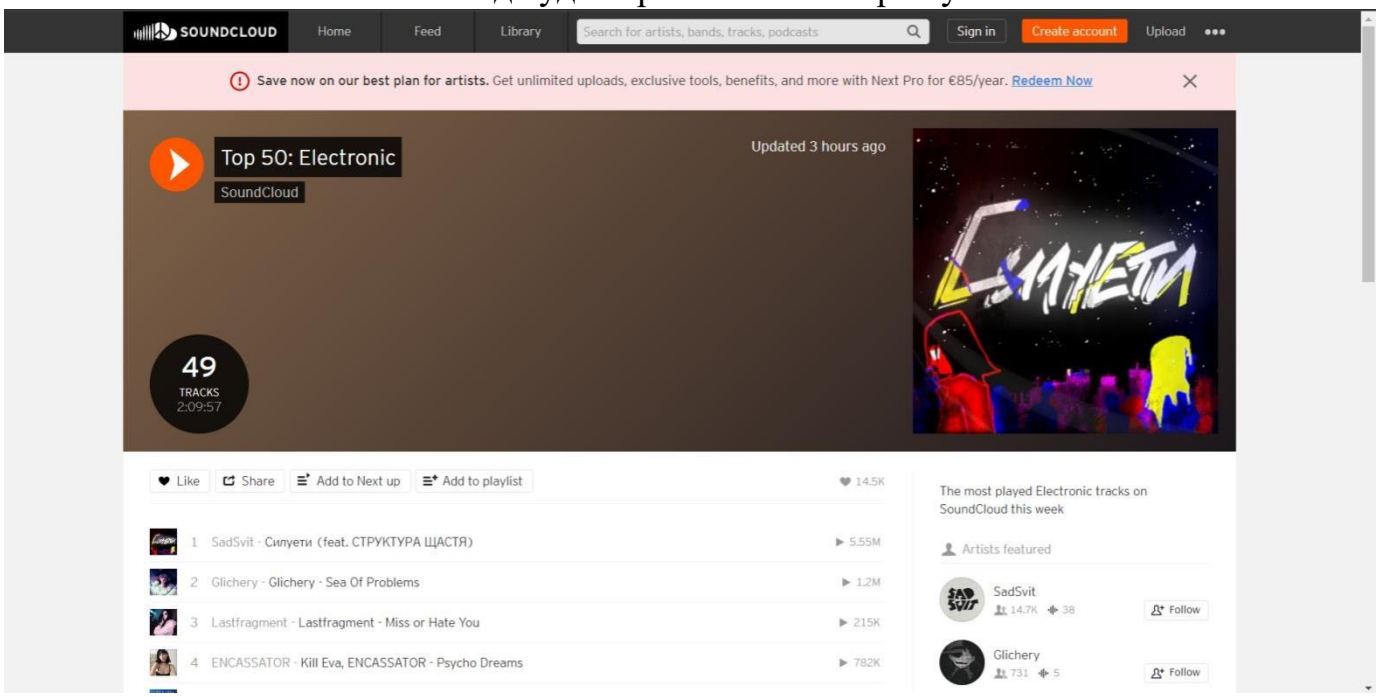


Рис. 1. 6. Вигляд аудіостримінгового сервісу ***Spotify***

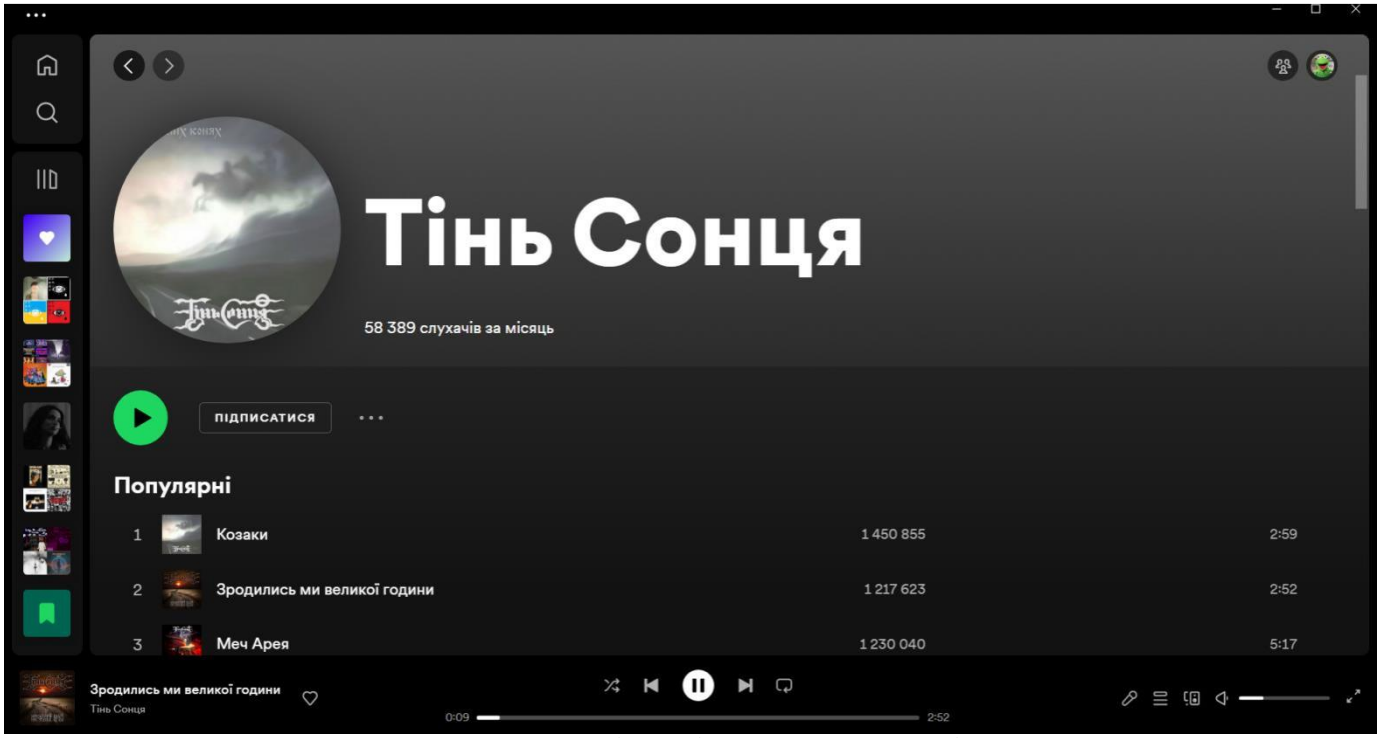
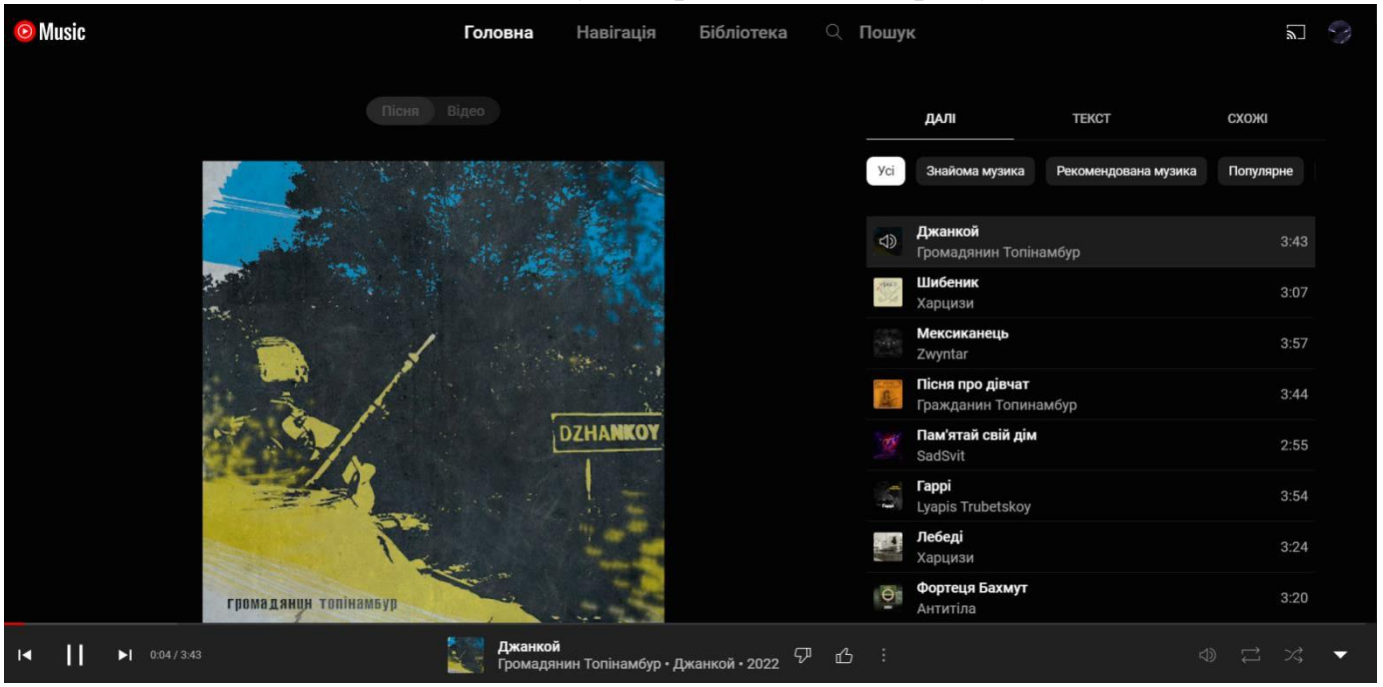


Рис. 1. 7. Вигляд аудіостримінгового сервісу ***Youtube Music***



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

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:

- **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.

- **Керованість:**

Інтерфейс користувача ПЗ має бути простим та зручним для використання незалежно від рівня технічної грамотності користувачів.

- **Швидкодія:**

Для максимально ефективної роботи з користувачем, необхідно, щоб програмне забезпечення працювало швидко та надійно, уникаючи зайвих затримок.

- **Документація**:

Документація до програмного забезпечення має бути належним чином підготовлена, щоб надати користувачам можливість правильно використовувати його та зрозуміти всі його функції та можливості.

- **Покращення**:

ПЗ повинно бути гнучким щодо можливості майбутньої модифікації шляхом додавання нових модулів або компонентів, щоб забезпечити його адаптацію до змінних потреб користувачів.

2.2. Розробка структурної схеми програми

Структурна схема програми складається з трьох основних блоків:

1. Блок взаємодії *клієнта*
    - *графічний інтерфейс* користувача;
2. Блок взаємодії *сервера*
    - *аудіо рушій;*
3. Блок взаємодії *посередника* між *клієнтом* та *сервером*
    - *аудіо програвач;*

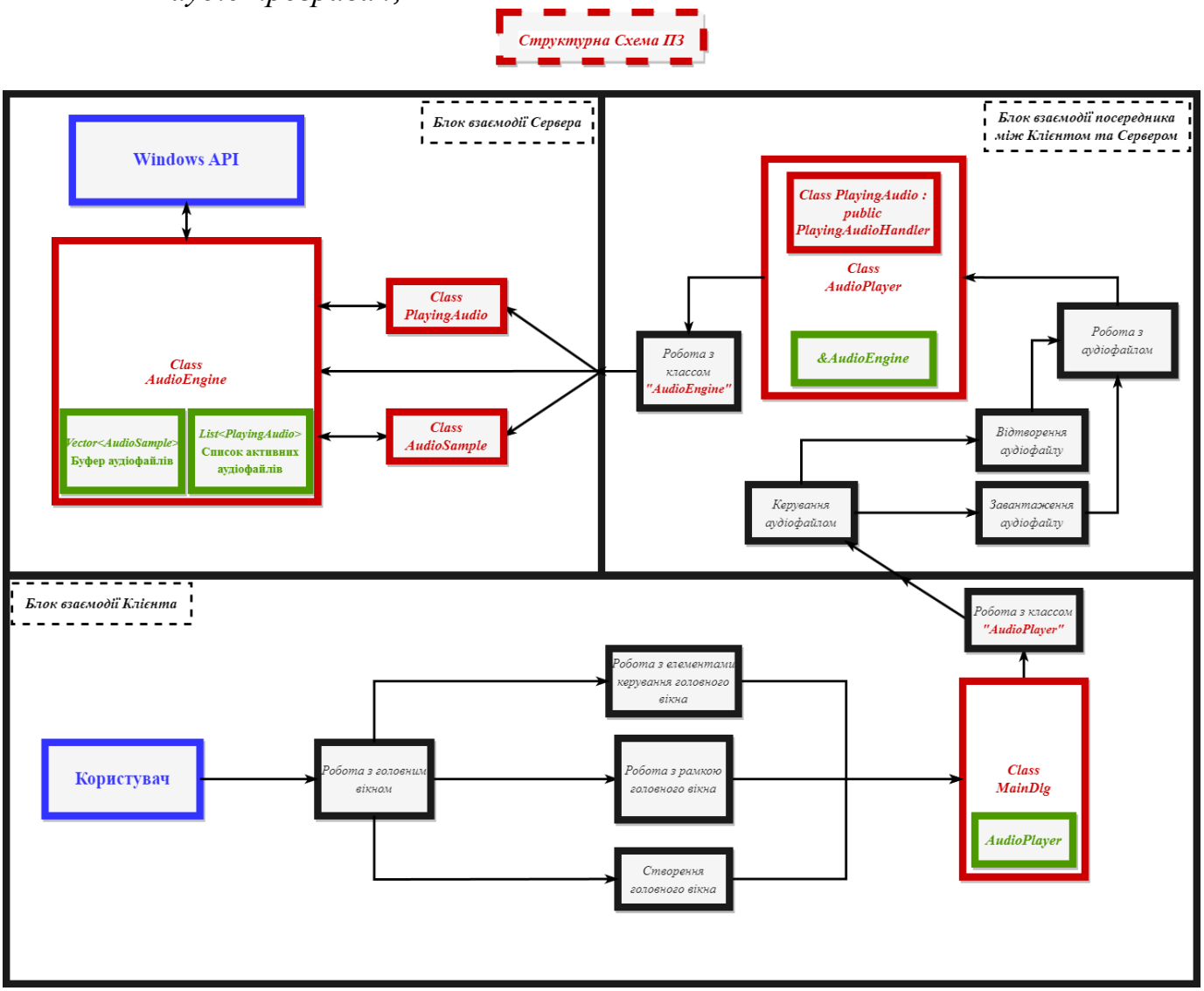


Рис 2.1. Структурна схема ПЗ

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*.

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 *C++* з використанням бібліотеки програмного інтерфейсу ОС *“Windows”* - *Visual Studio* має широкий 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”*.

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)
```

2. *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.

3. *Control* – абстрактний клас-інтерфейс. Використовується для реалізації класів з уніфікованим програмним інтерфейсом. Відповідає за створення та роботу елемента керування користувацького інтерфейсу.

4. *ControlButton* – відповідає за створення та роботу *кнопки*.

5. *ControlStaticText* – відповідає за створення та роботу *статичного тексту*.

6. *ControlCombobox* – відповідає за створення та роботу *викидного списку*.

7. *ControlSlider* – відповідає за створення та роботу *бігунка*.

4.1.1. Головне вікно програми

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

a. *Кнопка* згортання *головного вікна*. Дана кнопка зображена на **рис 3.1** під номером **0**.

b. *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**.

c. Шлях до файлу обраної композиції. Він зображений на **рис 3.1** під номером **2**.

d. *Викидний список* аудіокомпозицій(*плейлист*). Після вибору аудіокомпозиції, вона потрапляє у *викидний список*(*плейлист*). Даний викидний список зображений на **рис 3.1** під номером **3**.

e. *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**.

f. *Бігунок* зміни позиції виконання аудіокомпозиції. Використовується для прогортання позиції виконання аудіокомпозиції в реальному часі. Даний бігунок зображений на **рис 3.1** під номером **5**.

g. Теперішня позиція виконання в секундах/хвилинах/годинах (сек:хв:год). Віна зображена на **рис 3.1** під номером **6**.

h. *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**.

i. *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**.

j. *Кнопка* скидання вибраної композиції. Дана кнопка зображена на **рис 3.1** під номером **9**.

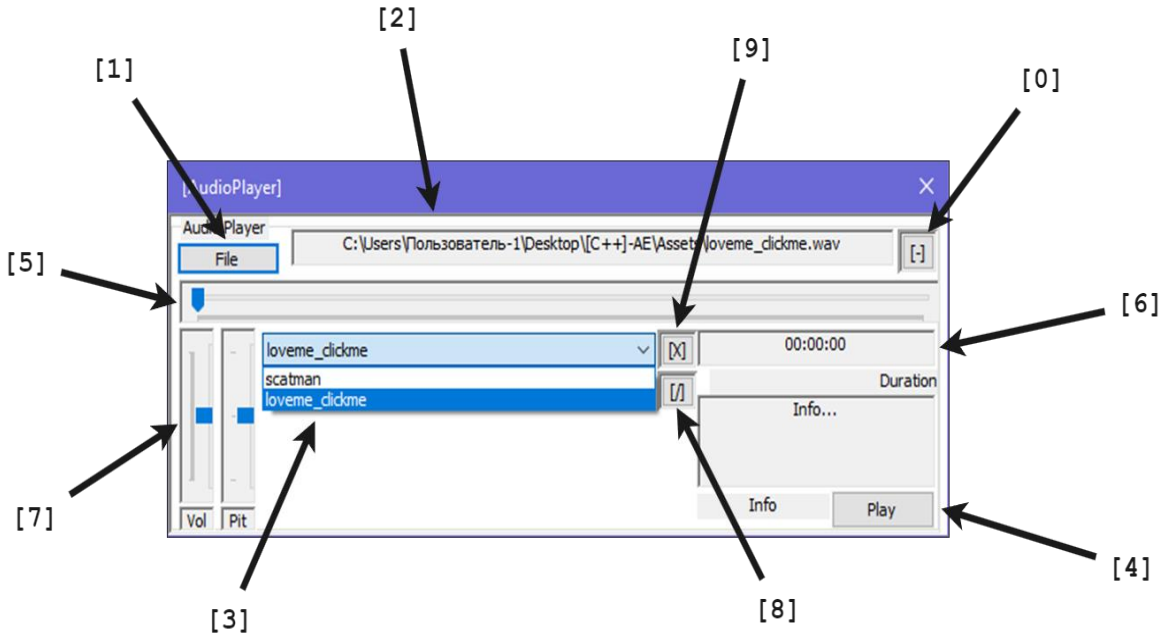


Рис 3.1. Інтерфейс *головного вікна*

3.2. Розробка модуля блока взаємодії сервера

У даному проекті було розроблено *3 класи*:

3.2.1. 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).

2. **`nBitsPerSample`** *(бітрейт)* – це кількість біт на одиницю аудіоданих. В даному випатку одиниця аудіоданих кодується 16 бітами, тобто 2 байтами.
3. **`nChannels`** *(кількість аудіоканалів)* – моно/стерео звук. В даному випадку використовуються монофонічні аудіодані (1 канал).

4. **`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.

5. **`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.

3.2.2. 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.

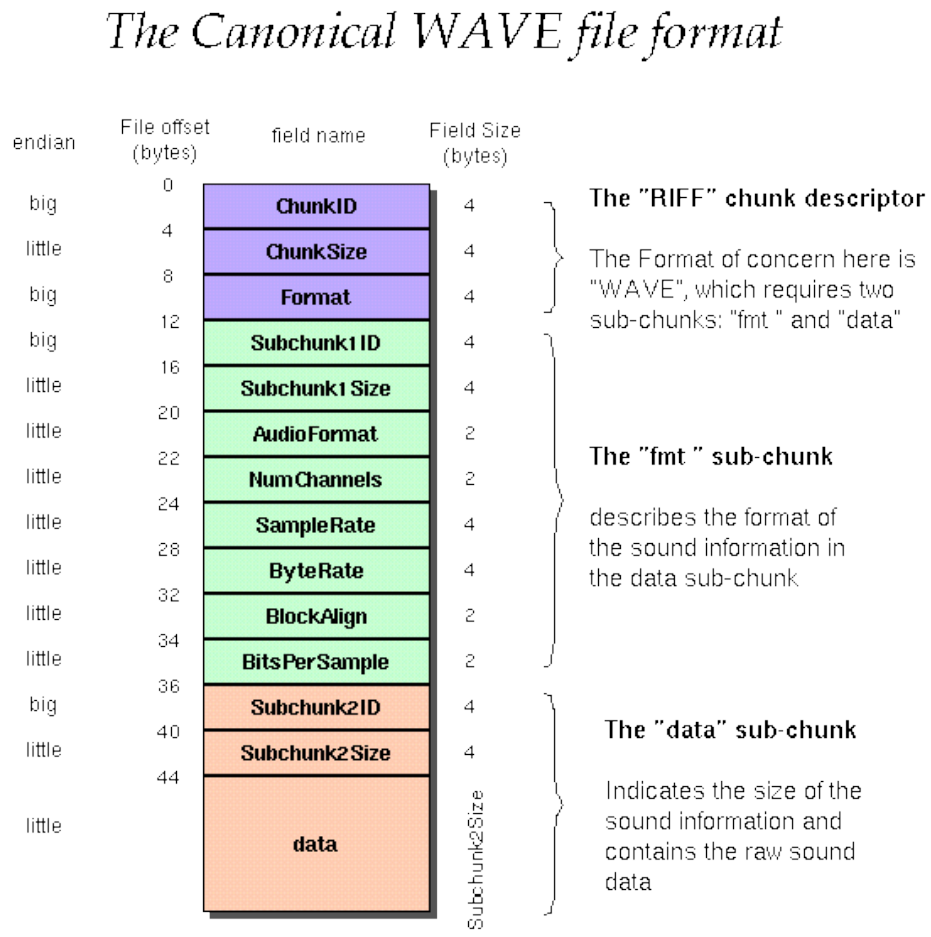


Рис. 3.2. Формат *.WAV* файлу

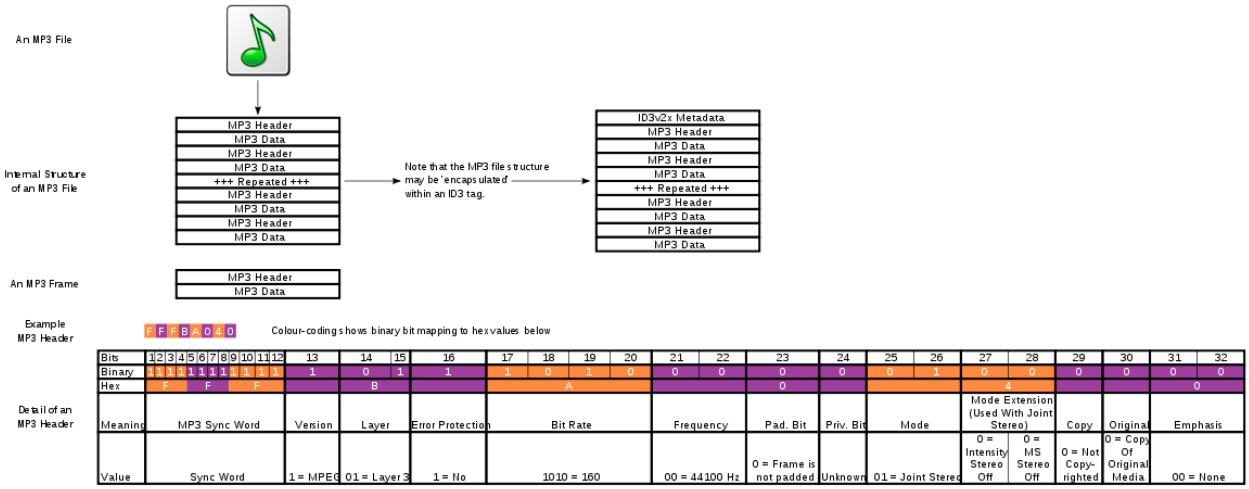


Рис. 3.3. Формат *.MP3* файлу

3.2.3. 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.

3.2.4. 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.

3.3. Розробка модуля взаємодії посередника між клієнтом та сервером

У даному проекті було розроблено *2 класи*:

3.3.1. 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*.

3.3.2. 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:*

3.3.2.1. 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
```

3.3.2.2. 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
```

1. – метод зміни *стану* виконання. Приймає *ідентифікатор стану* виконання, та встановлює його як теперішній.
2. – метод перемикання *стану* виконання. Перемикає *стан* виконання на протилежний.
3. – 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
```

1. – метод зміни *позиції* відтворення. Приймає *числове значення* в діапазоні від 0 до 1, та встановлює його як значення теперішньої позиції відтворення.
2. – метод отримання *теперішньої позиції* відтворення. Повертає *числове значення* в діапазоні від 0 до 1.

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
```

1. – метод зміни *гучності* відтворення. Приймає *числове значення* починаючи з 0, та встановлює його як значення теперішньої гучності відтворення.
2. – метод отримання *теперішньої гучності* відтворення. Повертає *числове значення* в починаючи з 0.
3. – метод зміни *швидкості* відтворення. Приймає *числове значення* починаючи з 0, та встановлює його як значення теперішньої швидкості відтворення.
4. – 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.

РОЗДІЛ 4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ

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).

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.

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 C++ 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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. [MSDN] Windows Multimedia [Електронний ресурс]: https://learn.microsoft.com/en-us/windows/win32/Multimedia/windows-multimedia-start-page
2. [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
3. [StackOverflow/Questions] Reading the Data of a .MP3 file [Електронний ресурс]: https://stackoverflow.com/questions/2968656/reading-mp3-files
4. [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
5. Євген Музиченко. Низькорівневе програмування звуку в ОС “Windows”. Журнал Комп’ютер Пресс #6-2000.
6. Юрій Щупак. Win32 API. Розробка програм для Windows. СПб.: Питер, 2008. - 592 с.: ил.

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);
```

```
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>
```

```
#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
```

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
```

```
//////////////////////////////////////////////////////////////////////////////
//
// 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
```

```
//////////////////////////////////////////////////////////////////////////////
//
// 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;
```

```
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;
```

```
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
) :
```

```
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;
```

```
// 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);
}
```

```
// 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)); }
```

```
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();
}
```

```
// 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;
```

```
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); }
```

```
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);
}
```

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); }
```

```
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);
```

```
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...*/
}
```

```
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_;
}
```

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;
```

```
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]
	);
}
```

```
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);
```

```
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
	);
}
```

```
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);
}
```

```
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);
```

```
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;
```

```
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; }
```

```
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);
}
```

```
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_
```

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__);
	}
```

```
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
{
```

```
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]);
```

```
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
```

```
);
	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
)
```

```
{
	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)
{
```

```
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) {
```

```
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());
                    }
              }
```

```
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 };
```

```
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;
```

```
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);
}
```

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”

```
#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
	);
}
```

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
	);
}
```

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)
```

```
{
	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”

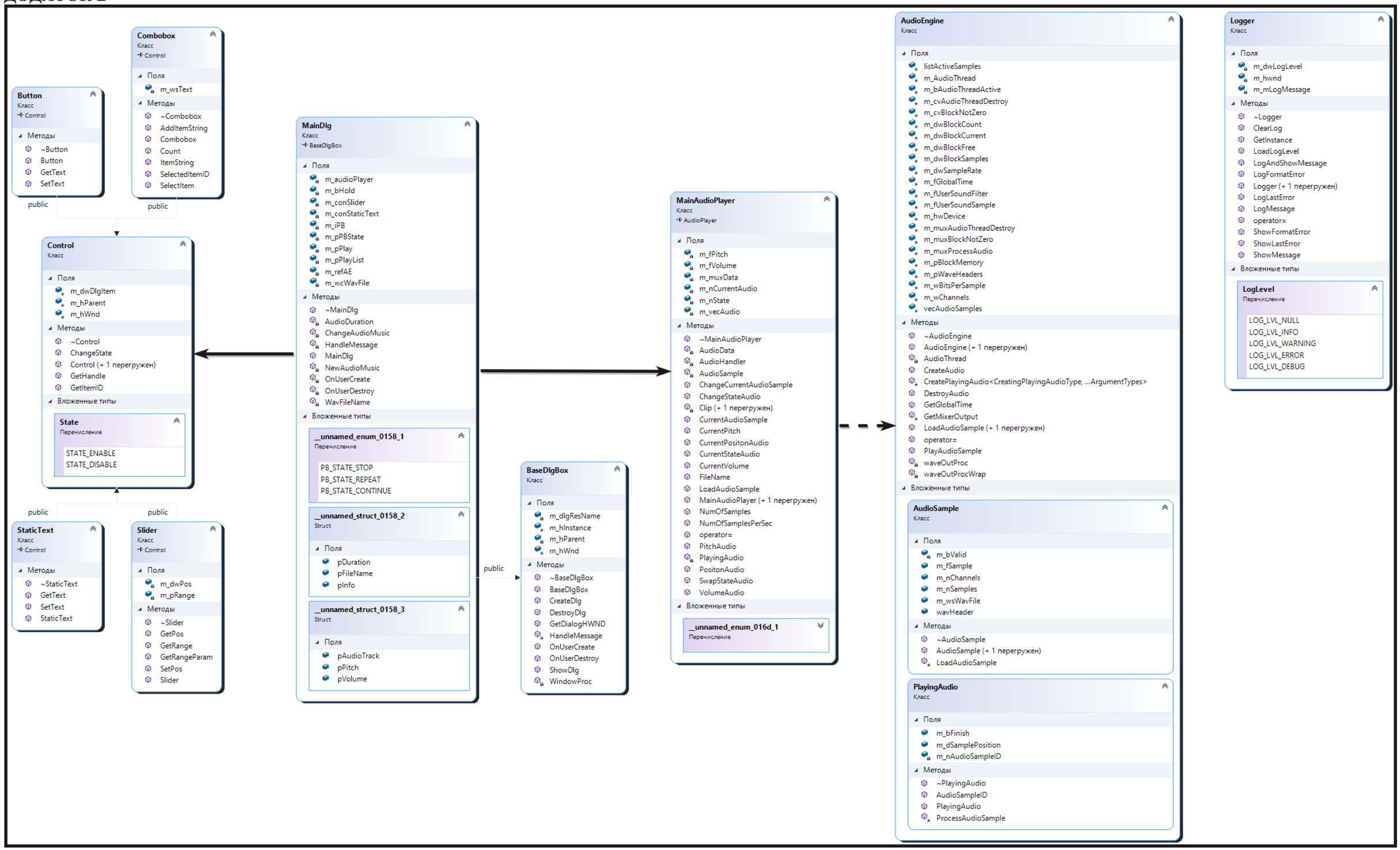
```
#pragma once
#ifndef _CONTROL_SLIDER_H_
#define _CONTROL_SLIDER_H_

#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);
```

```
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); }
```

ДОДАТОК Б


Button
Клас
Control
Методы
~Button
Button
GetText
SetText
Combobox
Клас
Control
Поля
m_wsText
Методы
~Combobox
AddItemString
Combobox
Count
ItemString
SelectedItemID
SelectItem
public
public
Control
Клас
Поля
m_dwDlgItem
m_hParent
m_hWnd
Методы
~Control
ChangeState
Control (+ 1 перегружен)
GetHandle
GetItemID
Вложенные типы
State
Перечисление
STATE_ENABLE
STATE_DISABLE
public
public
StaticText
Клас
Control
Методы
~StaticText
GetText
SetText
StaticText
Slider
Клас
Control
Поля
m_dwPos
m_pRange
Методы
~Slider
GetPos
GetRange
GetRangeParam
SetPos
Slider
MainDlg
Клас
BaseDlgBox
Поля
m_audioPlayer
m_bHold
m_conSlider
m_conStaticText
m_iPB
m_pPBState
m_pPlay
m_pPlayList
m_refAE
m_wcWavFile
Методы
~MainDlg
AudioDuration
ChangeAudioMusic
HandleMessage
MainDlg
NewAudioMusic
OnUserCreate
OnUserDestroy
WavFileName
Вложенные типы
__unnamed_enum_0158_1
Перечисление
PB_STATE_STOP
PB_STATE_REPEAT
PB_STATE_CONTINUE
__unnamed_struct_0158_2
Struct
Поля
pDuration
pFileName
pInfo
__unnamed_struct_0158_3
Struct
Поля
pAudioTrack
pPitch
pVolume
public
BaseDlgBox
Клас
Поля
m_dlgResName
m_hInstance
m_hParent
m_hWnd
Методы
~BaseDlgBox
BaseDlgBox
CreateDlg
DestroyDlg
GetDialogHWND
HandleMessage
OnUserCreate
OnUserDestroy
ShowDlg
WindowProc
MainAudioPlayer
Клас
AudioPlayer
Поля
m_fPitch
m_fVolume
m_muxData
m_nCurrentAudio
m_nState
m_vecAudio
Методы
~MainAudioPlayer
AudioData
AudioHandler
AudioSample
ChangeCurrentAudioSample
ChangeStateAudio
Clip (+ 1 перегружен)
CurrentAudioSample
CurrentPitch
CurrentPositonAudio
CurrentStateAudio
CurrentVolume
FileName
LoadAudioSample
MainAudioPlayer (+ 1 перегружен)
NumOfSamples
NumOfSamplesPerSec
operator=
PitchAudio
PlayingAudio
PositonAudio
SwapStateAudio
VolumeAudio
Вложенные типы
__unnamed_enum_016d_1
Перечисление
AudioEngine
Клас
Поля
listActiveSamples
m_AudioThread
m_bAudioThreadActive
m_cvAudioThreadDestroy
m_cvBlockNotZero
m_dwBlockCount
m_dwBlockCurrent
m_dwBlockFree
m_dwBlockSamples
m_dwSampleRate
m_fGlobalTime
m_fUserSoundFilter
m_fUserSoundSample
m_hwDevice
m_muxAudioThreadDestroy
m_muxBlockNotZero
m_muxProcessAudio
m_pBlockMemory
m_pWaveHeaders
m_wBitsPerSample
m_wChannels
vecAudioSamples
Методы
~AudioEngine
AudioEngine (+ 1 перегружен)
AudioThread
CreateAudio
CreatePlayingAudio<CreatingPlayingAudioType, ...ArgumentTypes>
DestroyAudio
GetGlobalTime
GetMixerOutput
LoadAudioSample (+ 1 перегружен)
operator=
PlayAudioSample
waveOutProc
waveOutProcWrap
Вложенные типы
AudioSample
Клас
Поля
m_bValid
m_fSample
m_nChannels
m_nSamples
m_wsWavFile
wavHeader
Методы
~AudioSample
AudioSample (+ 1 перегружен)
LoadAudioSample
PlayingAudio
Клас
Поля
m_bFinish
m_dSamplePosition
m_nAudioSampleID
Методы
~PlayingAudio
AudioSampleID
PlayingAudio
ProcessAudioSample
Logger
Клас
Поля
m_dwLogLevel
m_hwnd
m_mLogMessage
Методы
~Logger
ClearLog
GetInstance
LoadLogLevel
LogAndShowMessage
LogFormatError
Logger (+ 1 перегружен)
LogLastError
LogMessage
operator=
ShowFormatError
ShowLastError
ShowMessage
Вложенные типы
LogLevel
Перечисление
LOG_LVL_NULL
LOG_LVL_INFO
LOG_LVL_WARNING
LOG_LVL_ERROR
LOG_LVL_DEBUG