Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №1

на тему

Основы программирования в Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений.

Студент: гр.153502 Сидорова У.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc145669228)

[2 Теоретические сведения 3](#_Toc145669228)

[3 Результат работы программы 4](#_Toc145669229)

[Список использованных источников 6](#_Toc145669230)

[Приложение A 7](#_Toc145669231)

1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API*, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на *Win32 API*, таких как обработка оконных сообщений.

В качестве задачи необходимо разработать аудиопроигрыватель с возможностью воспроизведения различных аудиоформатов и создания плейлистов.

**2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

*Windows API* (также известный как *Win32 API, Windows Desktop API* и *Windows Classic AP*I) — это платформа на основе языка *C* для создания приложений *Windows*. Он существует с 1980-х годов и десятилетиями использовался для создания приложений *Windows*. Более продвинутые и простые в программировании платформы были построены на основе *Windows API*. Например, *MFC, ATL, .NET frameworks*. [1].

Для проигрывания аудиофайла и его управления использовалась функция *mciSendString.* Функция *mciSendString* отправляет командную строку устройству *MCI*. Устройство, на которое отправляется команда, указывается в командной строке [2].

Для обновления ползунка задавался таймер с помощью функции *SetTimer.* Она создает таймер с указанным значением времени ожидания [3].

Для изменения текущей позиции использоваласть команда *MCI\_SEEK.* Команда *MCI\_SEEK* изменяет текущую позицию в содержимом как можно быстрее. Видео- и аудиовыход отключается во время поиска. После завершения поиска устройство останавливается. *CD*-аудио, цифровое видео, видеомагнитофон, видеодиск и волновое аудио распознают эту команду [4].

3 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

При запуске программы открывается оконное приложение (рисунок 1).

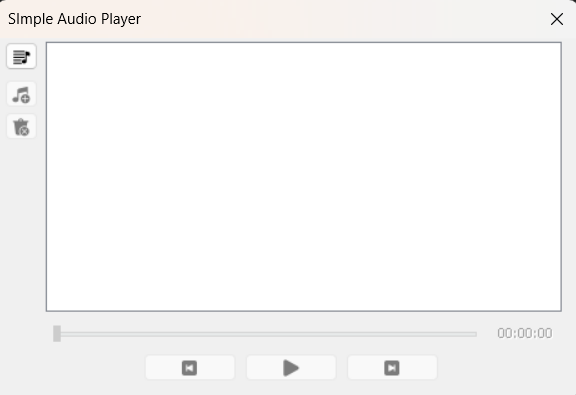


Рисунок 1 – Графический интерфейс программы

Функциональные возможности приложения:

– cоздать новый плейлист / открыть существующий плейлист;

При нажатии на верхнюю кнопку открывается файловый менеджер (рисунок 2), который позволяет открыть существующий файл с расширением .*playlist* или создать новый файл.

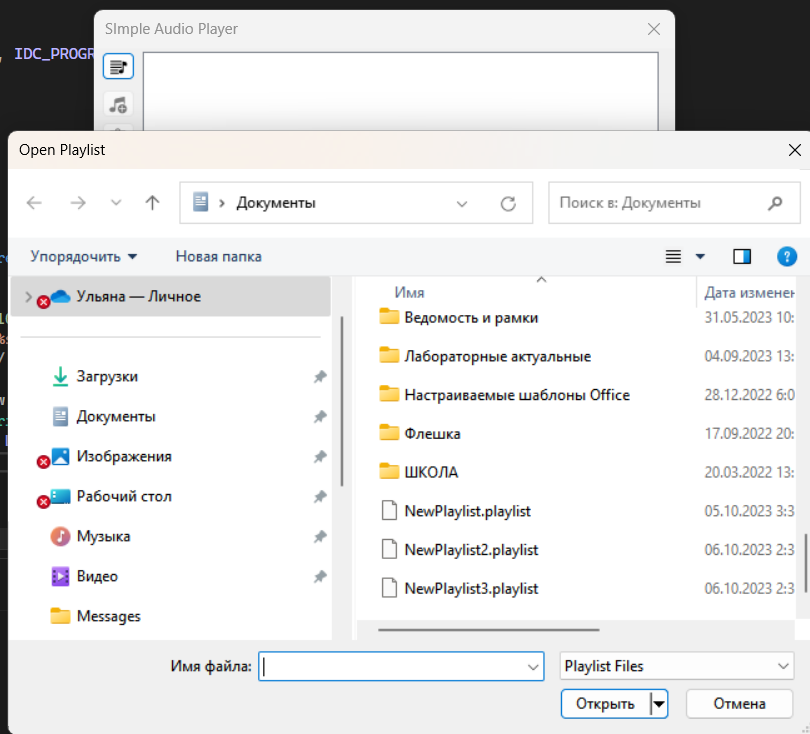


Рисунок 2 – Открытие файлового менеджера

– просмотр аудиофайлов, входящих в открытый плейлист;

После выбора текущего плейлиста появляется список аудиофайлов, входящих в плейлист (рисунок 3).

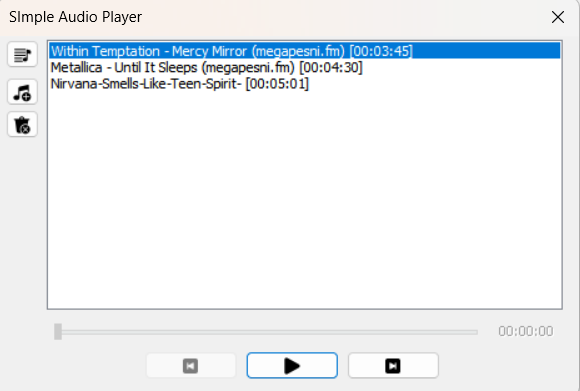


Рисунок 3 – Отображение доступных аудиофайлов

– навигация по открытому плейлисту;

При помощи кнопок со стрелками, которые направлены влево(предыдущий аудиофайл) и вправо(следующий аудиофайл) можно переключаться между аудиофайлами (рисунок 4). Если выбран первый аудиофайл, то кнопка влево становится неактивной, если последний – кнопка вправо.



Рисунок 4 – Кнопки навигации

– прослушивание / остановка аудиофайла;

При нажатии на кнопку *play*, воспроизводится выбранный аудиофайл, а кнопка play меняется на *pause*. При повторном нажатии проигрывание аудиофайла завершится, а кнопка вернется в первоначальное состояние (рисунок 5).

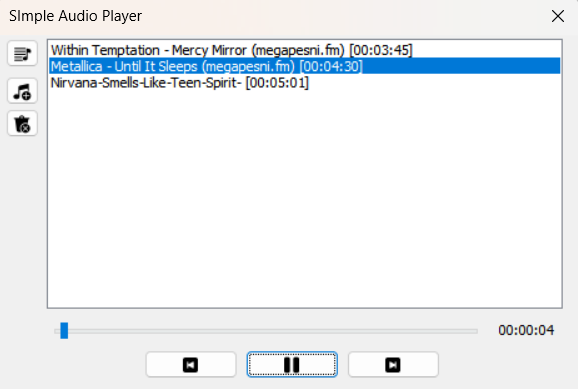


Рисунок 5 – Проигрывание аудиофайла

– перемотка аудиофайла на определенную позицию;

Над навигационной панелью расположен ползунок, рядом с которым отображается время проигрывания. Он становится активным при проигрывании аудиофайла (рисунок 6).



Рисунок 6 – Отображение ползунка и текущего времени

– добавление аудиофайла в плейлист;

Кнопка добавлении аудиофайла при невыбранном плейлисте неактивна. После выбора плейлиста она активируется и появляется возможность добавить аудиофайл в текущий плейлист. При нажатии открывается файловый менеджер, в котором отображаются файлы с расширениями .*mp3*, .*wav*, .*aiff*, .*wma*. После выбора аудиофайла он добавляется в плейлист.

– удаления аудиофайла из плейлиста;

Кнопка удаления аудиофайла при невыбранном плейлисте неактивна. После выбора плейлиста она активируется и появляется возможность удалить аудиофайл из текущего плейлиста. Для удаления необходимо выбрать аудиофайл, который надо удалить и нажать на кнопку удаления.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Walkthrough: Create a traditional Windows Desktop application (C++) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/cpp/windows/walkthrough-creating-windows-desktop-applications-cpp?view=msvc-170> – Дата доступа 02.10.2023.

[2] mciSendString function [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/previous-versions/dd757161(v=vs.85)> – Дата доступа 02.10.2023.

[3] SetTimer function (winuser.h) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-settimer> – Дата доступа 05.10.2023.

[4] MCI\_SEEK command [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/multimedia/mci-seek> – Дата доступа 02.10.2023.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл main.cpp**

#ifndef UNICODE

#define UNICODE

#endif

#include "MCIMediaPlayer.h"

#include "Helpers.h"

#include "resource.h"

#include <windows.h>

#include <CommCtrl.h>

#include <string>

static MCIMediaPlayer\* pMediaPlayer = NULL;

static HINSTANCE hAppInstance = NULL;

static HWND hMainWindow = NULL;

LRESULT CALLBACK MainDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

void UpdatePlayerControls();

void UpdatePlayPosition();

int WINAPI wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ PWSTR pCmdLine, \_In\_ int nCmdShow)

{

hAppInstance = hInstance;

HWND hMainDialog = ::CreateDialog(hInstance, MAKEINTRESOURCE(IDD\_MAINWND), NULL, MainDialogProc);

if (hMainDialog == NULL)

{

return -1;

}

hMainWindow = hMainDialog;

::ShowWindow(hMainDialog, nCmdShow);

::UpdateWindow(hMainDialog);

pMediaPlayer = new MCIMediaPlayer(hMainDialog);

UpdatePlayerControls();

MSG msg = { };

while (::GetMessage(&msg, NULL, 0, 0) > 0)

{

if (!IsDialogMessage(hMainDialog, &msg))

{

::TranslateMessage(&msg);

::DispatchMessage(&msg);

}

}

delete pMediaPlayer;

return 0;

}

INT\_PTR MainDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_DESTROY:

::PostQuitMessage(0);

return TRUE;

case WM\_CLOSE:

::DestroyWindow(hwnd);

return TRUE;

case WM\_TIMER:

if (wParam == IDT\_TIMER)

{

UpdatePlayPosition();

}

return TRUE;

case MM\_MCINOTIFY:

case MM\_MCISIGNAL:

pMediaPlayer->handleMCINotify(hwnd, uMsg, wParam, lParam);

return TRUE;

case WM\_HSCROLL:

{

int position = ::SendDlgItemMessage(hMainWindow, IDC\_PROGRESS\_SLIDER, TBM\_GETPOS, 0, 0);

pMediaPlayer->setPlayPostionInMs(position);

return TRUE;

}

break;

case MCIMediaPlayer::MCI\_MP\_TRACK\_ADDED:

{

int trackIndex = static\_cast<int>(wParam);

MCIMediaPlayer::TrackInfo\* trackInfo = reinterpret\_cast<MCIMediaPlayer::TrackInfo\*>(lParam);

char trackTitle[1024];

int trackLengthInSec = trackInfo->LengthInMs / 1000;

std::snprintf(trackTitle, sizeof(trackTitle), "%s [%02d:%02d:%02d]",

trackInfo->Title.c\_str(), trackLengthInSec / 3600, (trackLengthInSec % 3600) / 60, (trackLengthInSec % 3600) % 60);

LRESULT index = ::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_ADDSTRING, 0,

(LPARAM)Helpers::StringToWideString(std::string(trackTitle)).c\_str());

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_SETITEMDATA, index, (LPARAM)trackInfo);

UpdatePlayerControls();

}

break;

case MCIMediaPlayer::MCI\_MP\_TRACK\_REMOVED:

{

int trackIndex = static\_cast<int>(wParam);

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_DELETESTRING, trackIndex, 0);

UpdatePlayerControls();

}

break;

case MCIMediaPlayer::MCI\_MP\_PLAYLIST\_LOADED:

{

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_SETCURSEL, 0, 0);

UpdatePlayerControls();

}

break;

case MCIMediaPlayer::MCI\_MP\_PLAYLIST\_CLEAR:

{

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_RESETCONTENT, 0, 0);

UpdatePlayerControls();

}

break;

case MCIMediaPlayer::MCI\_MP\_PLAY\_TRACK:

{

int trackIndex = static\_cast<int>(wParam);

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_SETCURSEL, trackIndex, 0);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_PLAY\_BUTTON, IDI\_PAUSE\_ICON);

UpdatePlayerControls();

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_SETCURSEL, trackIndex, 0);

MCIMediaPlayer::TrackInfo const& trackInfo = pMediaPlayer->getTrackInfo(trackIndex);

::SendDlgItemMessage(hMainWindow, IDC\_PROGRESS\_SLIDER, TBM\_SETRANGEMIN, TRUE, 0);

::SendDlgItemMessage(hMainWindow, IDC\_PROGRESS\_SLIDER, TBM\_SETRANGEMAX, TRUE, trackInfo.LengthInMs);

::SendDlgItemMessage(hMainWindow, IDC\_PROGRESS\_SLIDER, TBM\_SETPAGESIZE, 0, 10000);

UpdatePlayPosition();

::SetTimer(hwnd, IDT\_TIMER, 100, NULL);

}

break;

case MCIMediaPlayer::MCI\_MP\_STOP\_TRACK:

case MCIMediaPlayer::MCI\_MP\_PAUSE\_TRACK:

{

::KillTimer(hwnd, IDT\_TIMER);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_PLAY\_BUTTON, IDI\_PLAY\_ICON);

UpdatePlayerControls();

}

break;

case MCIMediaPlayer::MCI\_MP\_TRACK\_CHANGED:

{

int trackIndex = static\_cast<int>(wParam);

::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_SETCURSEL, trackIndex, 0);

UpdatePlayerControls();

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDC\_OPEN\_PLAYLIST\_BUTTON:

{

std::string playListFilePath = Helpers::BrowseForFile(hAppInstance, hwnd, TEXT("Open Playlist"),

TEXT("Playlist Files\0\*.playlist\0"), TEXT("playlist"), false);

if (playListFilePath.empty())

{

return TRUE;

}

if (!pMediaPlayer->openPlayList(playListFilePath))

{

::MessageBox(hMainWindow, TEXT("Failed to open playlist"), TEXT("Open Playlist"), MB\_OK);

}

return TRUE;

}

break;

case IDC\_PLAY\_BUTTON:

if (pMediaPlayer->isPlaying())

{

pMediaPlayer->pause();

}

else

{

pMediaPlayer->play();

}

return TRUE;

case IDC\_PREV\_BUTTON:

pMediaPlayer->prev();

return TRUE;

case IDC\_NEXT\_BUTTON:

pMediaPlayer->next();

return TRUE;

case IDC\_PLAYLIST:

{

switch HIWORD(wParam)

{

case LBN\_SELCHANGE:

{

int trackIndex = static\_cast<int>(::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_GETCURSEL, 0, 0));

if (trackIndex != LB\_ERR)

{

pMediaPlayer->moveToTrack(trackIndex);

if (pMediaPlayer->isPlaying())

{

pMediaPlayer->play();

}

UpdatePlayerControls();

}

}

break;

case LBN\_DBLCLK:

{

int trackIndex = static\_cast<int>(::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_GETCURSEL, 0, 0));

if (trackIndex != LB\_ERR)

{

pMediaPlayer->moveToTrack(trackIndex);

pMediaPlayer->play();

UpdatePlayerControls();

}

}

break;

}

return TRUE;

}

break;

case IDC\_ADDSONG\_BUTTON:

{

std::string trackFilePath = Helpers::BrowseForFile(hAppInstance, hwnd, TEXT("Add Track"),

TEXT("Audio Files\0\*.mp3;\*.wav;\*.aiff;\*.wma\0"), NULL, true);

if (trackFilePath.empty())

{

return TRUE;

}

if (!pMediaPlayer->addTrackToPlayList(trackFilePath))

{

::MessageBox(hMainWindow, TEXT("Failed to add track"), TEXT("Add Track"), MB\_OK);

}

pMediaPlayer->savePlayList();

return TRUE;

}

break;

case IDC\_REMOVESONG\_BUTTON:

{

int trackIndex = static\_cast<int>(::SendDlgItemMessage(hMainWindow, IDC\_PLAYLIST, LB\_GETCURSEL, 0, 0));

if (trackIndex != LB\_ERR)

{

pMediaPlayer->removeTrackFromPlayList(trackIndex);

pMediaPlayer->savePlayList();

}

return TRUE;

}

break;

}

break;

case WM\_INITDIALOG:

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_NEW\_PLAYLIST\_BUTTON, IDI\_NEW\_PLAYLIST\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_OPEN\_PLAYLIST\_BUTTON, IDI\_OPEN\_PLAYLIST\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_SAVE\_PLAYLIST\_BUTTON, IDI\_SAVE\_PLAYLIST\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_ADDSONG\_BUTTON, IDI\_ADDSONG\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_REMOVESONG\_BUTTON, IDI\_REMOVESONG\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_PLAY\_BUTTON, IDI\_PLAY\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_NEXT\_BUTTON, IDI\_NEXT\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_PREV\_BUTTON, IDI\_PREV\_ICON);

return TRUE;

}

return FALSE;

}

void UpdatePlayerControls()

{

Helpers::SetControlEnabled(hMainWindow, IDC\_ADDSONG\_BUTTON, pMediaPlayer->isPlaylistLoaded());

Helpers::SetControlEnabled(hMainWindow, IDC\_REMOVESONG\_BUTTON, pMediaPlayer->isPlaylistLoaded());

Helpers::SetControlEnabled(hMainWindow, IDC\_PLAYLIST, pMediaPlayer->isPlaylistLoaded());

Helpers::SetControlEnabled(hMainWindow, IDC\_PROGRESS\_SLIDER, pMediaPlayer->isPlaying());

Helpers::SetControlEnabled(hMainWindow, IDC\_CURRENT\_TIME\_STATIC, pMediaPlayer->isPlaying());

Helpers::SetControlEnabled(hMainWindow, IDC\_PLAY\_BUTTON, pMediaPlayer->trackCount() > 0);

Helpers::SetControlEnabled(hMainWindow, IDC\_PREV\_BUTTON, pMediaPlayer->hasPrev());

Helpers::SetControlEnabled(hMainWindow, IDC\_NEXT\_BUTTON, pMediaPlayer->hasNext());

UpdatePlayPosition();

}

void UpdatePlayPosition()

{

unsigned long long position = pMediaPlayer->getPlayPostionInMs();

::SendDlgItemMessage(hMainWindow, IDC\_PROGRESS\_SLIDER, TBM\_SETPOS, TRUE, position);

position /= 1000;

char positionTime[64];

std::snprintf(positionTime, sizeof(positionTime), "%02d:%02d:%02d", position / 3600, (position % 3600) / 60, (position % 3600) % 60);

::SendDlgItemMessage(hMainWindow, IDC\_CURRENT\_TIME\_STATIC, WM\_SETTEXT, NULL, (LPARAM)Helpers::StringToWideString(positionTime).c\_str());

}

**Файл MCIMediaPlayer.h**

#pragma once

#include <string>

#include <vector>

#include <windows.h>

#include <mciapi.h>

class MCIMediaPlayer

{

public:

struct TrackInfo

{

MCIDEVICEID MciDeviceId;

std::string Uri;

std::string Title;

std::string Artist;

unsigned long long LengthInMs;

};

typedef std::vector<TrackInfo> TrackList;

enum HWNDMessage

{

MCI\_MP\_PLAYLIST\_LOADED = WM\_APP + 1,

MCI\_MP\_PLAYLIST\_SAVED,

MCI\_MP\_PLAYLIST\_CLEAR,

MCI\_MP\_TRACK\_ADDED,

MCI\_MP\_TRACK\_REMOVED,

MCI\_MP\_PLAY\_TRACK,

MCI\_MP\_PAUSE\_TRACK,

MCI\_MP\_STOP\_TRACK,

MCI\_MP\_TRACK\_CHANGED

};

MCIMediaPlayer(HWND mciNotifyHandle);

bool play();

bool pause();

bool stop();

bool next();

bool prev();

bool moveToTrack(int trackIndex);

bool hasNext() const;

bool hasPrev() const;

bool isPlaylistLoaded() const;

bool isPlaying() const;

bool openPlayList(std::string const& filePath);

bool savePlayList();

bool savePlayListAs(std::string const& filePath);

bool addTrackToPlayList(std::string const& uri);

bool removeTrackFromPlayList(int trackIndex);

TrackInfo const& getTrackInfo(int trackIndex) const;

TrackList::const\_iterator beginTrack() const;

TrackList::const\_iterator endTrack() const;

int trackCount() const;

unsigned long long getPlayPostionInMs() const;

bool setPlayPostionInMs(unsigned long long position);

LRESULT handleMCINotify(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

private:

bool mciOpenMedia(std::string const& uri, TrackInfo& trackInfo);

bool mciPlayMedia(TrackInfo& trackInfo);

bool mciPauseMedia(TrackInfo& trackInfo);

bool mciCloseMedia(TrackInfo& trackInfo);

bool addTrackPrivate(std::string const& uri);

void clearPlayList();

void notifyWindow(HWNDMessage msg);

void notifyWindow(HWNDMessage msg,WPARAM wParam);

void notifyWindow(HWNDMessage msg, WPARAM wParam, LPARAM lParam);

private:

enum Status

{

PLAY,

STOP,

PAUSE

};

Status \_currentStatus;

TrackList \_trackList;

int \_currentTrackIndex;

std::string \_playListFilePath;

HWND \_notifyHandle;

};

**Файл MCIMediaPlayer.cpp**

#include "MCIMediaPlayer.h"

#include "Helpers.h"

#ifndef UNICODE

#define UNICODE

#endif

#include <cassert>

#include <fstream>

MCIMediaPlayer::MCIMediaPlayer(HWND mciNotifyHandle) :

\_currentStatus(STOP),

\_trackList(),

\_currentTrackIndex(-1),

\_playListFilePath(),

\_notifyHandle(mciNotifyHandle)

{

}

bool MCIMediaPlayer::play()

{

if (\_currentTrackIndex < 0)

{

return false;

}

TrackInfo& trackInfo = \_trackList[\_currentTrackIndex];

if (mciPlayMedia(trackInfo))

{

\_currentStatus = PLAY;

notifyWindow(MCI\_MP\_PLAY\_TRACK, \_currentTrackIndex);

return true;

}

return false;

}

bool MCIMediaPlayer::stop()

{

if (\_currentTrackIndex < 0)

{

return false;

}

TrackInfo& trackInfo = \_trackList[\_currentTrackIndex];

if (mciCloseMedia(trackInfo))

{

\_currentStatus = STOP;

notifyWindow(MCI\_MP\_STOP\_TRACK);

return true;

}

return false;

}

bool MCIMediaPlayer::pause()

{

if (\_currentTrackIndex < 0)

{

return false;

}

TrackInfo& trackInfo = \_trackList[\_currentTrackIndex];

if (mciPauseMedia(trackInfo))

{

\_currentStatus = PAUSE;

notifyWindow(MCI\_MP\_PAUSE\_TRACK);

return true;

}

return false;;

}

bool MCIMediaPlayer::next()

{

if (!hasNext())

{

return false;

}

moveToTrack(\_currentTrackIndex + 1);

if (isPlaying())

{

play();

}

return true;

}

bool MCIMediaPlayer::prev()

{

if (!hasPrev())

{

return false;

}

moveToTrack(\_currentTrackIndex - 1);

if (isPlaying())

{

play();

}

return true;

}

bool MCIMediaPlayer::moveToTrack(int trackIndex)

{

if (trackIndex < 0 || trackIndex >= \_trackList.size())

{

return false;

}

if (isPlaying() && \_currentTrackIndex != -1)

{

mciCloseMedia(\_trackList[\_currentTrackIndex]);

}

\_currentTrackIndex = trackIndex;

notifyWindow(MCI\_MP\_TRACK\_CHANGED, \_currentTrackIndex);

return true;

}

bool MCIMediaPlayer::hasNext() const

{

if (\_currentTrackIndex + 1 >= \_trackList.size())

{

return false;

}

return true;

}

bool MCIMediaPlayer::hasPrev() const

{

if (\_currentTrackIndex - 1 < 0)

{

return false;

}

return true;

}

bool MCIMediaPlayer::isPlaylistLoaded() const

{

return !\_playListFilePath.empty();

}

bool MCIMediaPlayer::isPlaying() const

{

return \_currentStatus == PLAY;

}

bool MCIMediaPlayer::openPlayList(std::string const& filePath)

{

std::ifstream playListStream(filePath);

std::string nextTrackPath;

stop();

clearPlayList();

while (std::getline(playListStream, nextTrackPath))

{

if (nextTrackPath.empty())

{

continue;

}

addTrackPrivate(nextTrackPath);

}

\_playListFilePath = filePath;

\_currentTrackIndex = \_trackList.empty() ? -1 : 0;

notifyWindow(MCI\_MP\_PLAYLIST\_LOADED);

return true;

}

bool MCIMediaPlayer::savePlayList()

{

if (\_playListFilePath.empty())

{

return false;

}

std::ofstream playListStream(\_playListFilePath, std::ofstream::out | std::ofstream::trunc);

for(auto nextTrackInfo : \_trackList)

{

playListStream << nextTrackInfo.Uri << std::endl;

}

notifyWindow(MCI\_MP\_PLAYLIST\_SAVED);

return true;

}

bool MCIMediaPlayer::savePlayListAs(std::string const& filePath)

{

return false;

}

bool MCIMediaPlayer::addTrackToPlayList(std::string const& uri)

{

assert(!\_playListFilePath.empty());

return addTrackPrivate(uri);

}

bool MCIMediaPlayer::removeTrackFromPlayList(int trackIndex)

{

assert(trackIndex >= 0 && trackIndex < \_trackList.size());

if (trackIndex == \_currentTrackIndex && isPlaying())

{

stop();

}

\_trackList.erase(std::next(\_trackList.begin(), trackIndex));

notifyWindow(MCI\_MP\_TRACK\_REMOVED, trackIndex);

if (trackIndex < \_trackList.size())

{

moveToTrack(trackIndex);

}

else if (!\_trackList.empty())

{

moveToTrack(trackIndex - 1);

}

return false;

}

bool MCIMediaPlayer::mciOpenMedia(std::string const& uri, TrackInfo& trackInfo)

{

std::wstring wUri = Helpers::StringToWideString(uri);

MCI\_OPEN\_PARMS mciOpenParams = { };

mciOpenParams.lpstrDeviceType = L"mpegvideo";

mciOpenParams.lpstrElementName = wUri.c\_str();

MCIERROR mciErroCode = mciSendCommand(0, MCI\_OPEN,

MCI\_OPEN\_TYPE | MCI\_OPEN\_ELEMENT, (LONG\_PTR)(LPMCI\_OPEN\_PARMS)&mciOpenParams);

if (mciErroCode != 0)

{

return false;

}

trackInfo.Uri = uri;

trackInfo.MciDeviceId = mciOpenParams.wDeviceID;

MCI\_SET\_PARMS mciSet = { NULL, MCI\_FORMAT\_MILLISECONDS, 0 };

mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_SET, (DWORD)MCI\_SET\_TIME\_FORMAT,

(LONG\_PTR)(LPMCI\_SET\_PARMS)&mciSet);

if (mciErroCode != 0)

{

mciCloseMedia(trackInfo);

return false;

}

MCI\_STATUS\_PARMS mciStatus = { NULL, 0, MCI\_STATUS\_LENGTH, 1 };

mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_STATUS, (DWORD)MCI\_STATUS\_ITEM,

(LONG\_PTR)(LPMCI\_STATUS\_PARMS)&mciStatus);

if (mciErroCode == 0)

{

trackInfo.LengthInMs = mciStatus.dwReturn;

}

return true;

}

bool MCIMediaPlayer::mciPlayMedia(TrackInfo& trackInfo)

{

if (trackInfo.MciDeviceId == 0 && !mciOpenMedia(trackInfo.Uri, trackInfo))

{

return false;

}

MCI\_PLAY\_PARMS mciPlayParams = { };

mciPlayParams.dwCallback = (LONG\_PTR)\_notifyHandle;

MCIERROR mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_PLAY, MCI\_NOTIFY,

(LONG\_PTR)(LPMCI\_PLAY\_PARMS)&mciPlayParams);

if (mciErroCode != 0)

{

return false;

}

return true;

}

bool MCIMediaPlayer::mciPauseMedia(TrackInfo& trackInfo)

{

MCI\_GENERIC\_PARMS mciGeneric;

MCIERROR mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_PAUSE, MCI\_WAIT,

(LONG\_PTR)(LPMCI\_GENERIC\_PARMS)&mciGeneric);

if (mciErroCode != 0)

{

return false;

}

return true;

}

bool MCIMediaPlayer::mciCloseMedia(TrackInfo& trackInfo)

{

MCI\_GENERIC\_PARMS mciGeneric;

MCIERROR mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_CLOSE, MCI\_WAIT,

(LONG\_PTR)(LPMCI\_GENERIC\_PARMS)&mciGeneric);

if (mciErroCode != 0)

{

return false;

}

trackInfo.MciDeviceId = 0;

return true;

}

MCIMediaPlayer::TrackInfo const& MCIMediaPlayer::getTrackInfo(int trackIndex) const

{

assert(trackIndex >= 0 && trackIndex < \_trackList.size());

return \_trackList[trackIndex];

}

MCIMediaPlayer::TrackList::const\_iterator MCIMediaPlayer::beginTrack() const

{

return \_trackList.begin();

}

MCIMediaPlayer::TrackList::const\_iterator MCIMediaPlayer::endTrack() const

{

return \_trackList.end();

}

int MCIMediaPlayer::trackCount() const

{

return static\_cast<int>(\_trackList.size());

}

unsigned long long MCIMediaPlayer::getPlayPostionInMs() const

{

if (!isPlaying())

{

return 0;

}

unsigned long long position = 0;

TrackInfo const& trackInfo = \_trackList[\_currentTrackIndex];

MCI\_STATUS\_PARMS mciStatus;

mciStatus.dwTrack = 1;

mciStatus.dwCallback = NULL;

mciStatus.dwItem = MCI\_STATUS\_POSITION;

MCIERROR mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_STATUS, (DWORD)MCI\_STATUS\_ITEM,

(LONG\_PTR)(LPMCI\_STATUS\_PARMS)&mciStatus);

if (mciErroCode == 0)

{

return mciStatus.dwReturn;

}

return 0;

}

bool MCIMediaPlayer::setPlayPostionInMs(unsigned long long position)

{

if (!isPlaying())

{

return false;

}

TrackInfo const& trackInfo = \_trackList[\_currentTrackIndex];

MCI\_SEEK\_PARMS mciSeek = { NULL, position };

MCIERROR mciErroCode = mciSendCommand(trackInfo.MciDeviceId, MCI\_SEEK, MCI\_WAIT | MCI\_TO,

(LONG\_PTR)(LPMCI\_SEEK\_PARMS)&mciSeek);

if (mciErroCode != 0)

{

return false;

}

play();

return true;

}

LRESULT MCIMediaPlayer::handleMCINotify(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

assert(hwnd == \_notifyHandle);

switch (uMsg)

{

case MM\_MCINOTIFY:

if (wParam == MCI\_NOTIFY\_SUCCESSFUL)

{

if (hasNext())

{

next();

}

else

{

stop();

}

}

break;

case MM\_MCISIGNAL:

break;

}

return 0;

}

bool MCIMediaPlayer::addTrackPrivate(std::string const& uri)

{

std::string baseFilename = uri.substr(uri.find\_last\_of("/\\") + 1);

std::string::size\_type const p(baseFilename.find\_last\_of('.'));

std::string trackName = baseFilename.substr(0, p);

TrackInfo trackInfo = { 0, uri, trackName, std::string(), 0 };

if (mciOpenMedia(uri, trackInfo))

{

mciCloseMedia(trackInfo);

\_trackList.push\_back(trackInfo);

notifyWindow(MCI\_MP\_TRACK\_ADDED, \_trackList.size() - 1, reinterpret\_cast<LPARAM>(&\_trackList.back()));

return true;

}

return false;

}

void MCIMediaPlayer::clearPlayList()

{

notifyWindow(MCI\_MP\_PLAYLIST\_CLEAR);

\_trackList.clear();

\_currentTrackIndex = -1;

}

void MCIMediaPlayer::notifyWindow(HWNDMessage msg)

{

notifyWindow(msg, 0);

}

void MCIMediaPlayer::notifyWindow(HWNDMessage msg, WPARAM wParam)

{

notifyWindow(msg, wParam, 0);

}

void MCIMediaPlayer::notifyWindow(HWNDMessage msg, WPARAM wParam, LPARAM lParam)

{

::SendMessage(\_notifyHandle, (UINT)msg, wParam, lParam);

}

**Файл Helpers.h**

#pragma once

#include <windows.h>

#include <string>

namespace Helpers

{

std::string WideStringToString(std::wstring const& wideStr);

std::wstring StringToWideString(std::string const& str);

bool SetButtonIcon(HINSTANCE hInstance, HWND hDlg, int buttonID, int iconID);

std::string BrowseForFile(HINSTANCE hInstance, HWND hParentWindow, LPCTSTR title,

LPCTSTR filter, LPCTSTR defExtention, bool existingOnly);

void SetControlEnabled(HWND hDlg, int controlID, bool enabled);

}

**Файл Helpers.cpp**

#include "Helpers.h"

#include <locale>

#include <codecvt>

namespace Helpers

{

bool SetButtonIcon(HINSTANCE hInstance, HWND hDlg, int buttonID, int iconID)

{

HICON hIcon = static\_cast<HICON>(::LoadImage(hInstance, MAKEINTRESOURCE(iconID), IMAGE\_ICON,

16, 16, LR\_DEFAULTCOLOR));

if (hIcon == NULL)

{

return FALSE;

}

::SendDlgItemMessage(hDlg, buttonID, BM\_SETIMAGE, (WPARAM)IMAGE\_ICON, (LPARAM)hIcon);

return TRUE;

}

std::string BrowseForFile(HINSTANCE hInstance, HWND hParentWindow, LPCTSTR title,

LPCTSTR filter, LPCTSTR defExtention, bool existingOnly)

{

static const int MaxFilePath = 1024;

OPENFILENAME of = {};

of.lStructSize = sizeof(OPENFILENAME);

of.hwndOwner = hParentWindow;

of.hInstance = hInstance;

of.lpstrFilter = filter;

of.lpstrDefExt = defExtention;

of.lpstrTitle = title;

of.Flags = OFN\_EXPLORER;

if (existingOnly)

{

of.Flags |= OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

}

TCHAR selectedFilePath[MaxFilePath] = { 0 };

of.lpstrFile = selectedFilePath;

of.nMaxFile = MaxFilePath;

BOOL ok = ::GetOpenFileName(&of);

if (ok)

{

std::wstring fileName(selectedFilePath);

return WideStringToString(fileName);

}

return std::string();

}

std::string WideStringToString(std::wstring const& wideStr)

{

std::wstring\_convert<std::codecvt\_utf8<wchar\_t>, wchar\_t> converter;

return converter.to\_bytes(wideStr);

}

std::wstring StringToWideString(std::string const& str)

{

std::wstring\_convert<std::codecvt\_utf8<wchar\_t>, wchar\_t> converter;

return converter.from\_bytes(str);

}

void SetControlEnabled(HWND hDlg, int controlID, bool enabled)

{

:: EnableWindow(GetDlgItem(hDlg, controlID), enabled);

}

}