#include <windows.h>

#include <mmsystem.h>

#include <strmif.h>

#include <control.h>

#pragma comment(lib, "strmiids.lib")

class **Mp3**

{

public:

**Mp3**();

~**Mp3**();

bool Load(LPCWSTR filename);

void Cleanup();

bool Play();

bool Pause();

bool Stop();

*// Poll this function with msTimeout = 0, so that it return immediately.*

*// If the* ***mp3*** *finished playing, WaitForCompletion will return true;*

bool WaitForCompletion(long msTimeout, long\* EvCode);

*// -10000 is lowest volume and 0 is highest volume, positive value > 0 will fail*

bool SetVolume(long vol);

*// -10000 is lowest volume and 0 is highest volume*

long GetVolume();

*// Returns the duration in 1/10 millionth of a second,*

*// meaning 10,000,000 == 1 second*

*// You have to divide the result by 10,000,000*

*// to get the duration in seconds.*

\_\_int64 GetDuration();

*// Returns the current playing position*

*// in 1/10 millionth of a second,*

*// meaning 10,000,000 == 1 second*

*// You have to divide the result by 10,000,000*

*// to get the duration in seconds.*

\_\_int64 GetCurrentPosition();

*// Seek to position with pCurrent and pStop*

*// bAbsolutePositioning specifies absolute or relative positioning.*

*// If pCurrent and pStop have the same value, the player will seek to the position*

*// and stop playing. Note: Even if pCurrent and pStop have the same value,*

*// avoid putting the same pointer into both of them, meaning put different*

*// pointers with the same dereferenced value.*

bool SetPositions(\_\_int64\* pCurrent, \_\_int64\* pStop, bool bAbsolutePositioning);

private:

IGraphBuilder \* pigb;

IMediaControl \* pimc;

IMediaEventEx \* pimex;

IBasicAudio \* piba;

IMediaSeeking \* pims;

bool ready;

*// Duration of the* ***MP3****.*

\_\_int64 duration;

};

#include "**Mp3**.h"

#include <uuids.h>

**Mp3**::**Mp3**()

{

pigb = NULL;

pimc = NULL;

pimex = NULL;

piba = NULL;

pims = NULL;

ready = false;

duration = 0;

}

**Mp3**::~**Mp3**()

{

Cleanup();

}

void **Mp3**::Cleanup()

{

if (pimc)

pimc->Stop();

if(pigb)

{

pigb->Release();

pigb = NULL;

}

if(pimc)

{

pimc->Release();

pimc = NULL;

}

if(pimex)

{

pimex->Release();

pimex = NULL;

}

if(piba)

{

piba->Release();

piba = NULL;

}

if(pims)

{

pims->Release();

pims = NULL;

}

ready = false;

}

bool **Mp3**::Load(LPCWSTR szFile)

{

Cleanup();

ready = false;

if (SUCCEEDED(CoCreateInstance( CLSID\_FilterGraph,

NULL,

CLSCTX\_INPROC\_SERVER,

IID\_IGraphBuilder,

(void \*\*)&this->pigb)))

{

pigb->QueryInterface(IID\_IMediaControl, (void \*\*)&pimc);

pigb->QueryInterface(IID\_IMediaEventEx, (void \*\*)&pimex);

pigb->QueryInterface(IID\_IBasicAudio, (void\*\*)&piba);

pigb->QueryInterface(IID\_IMediaSeeking, (void\*\*)&pims);

HRESULT hr = pigb->RenderFile(szFile, NULL);

if (SUCCEEDED(hr))

{

ready = true;

if(pims)

{

pims->SetTimeFormat(&TIME\_FORMAT\_MEDIA\_TIME);

pims->GetDuration(&duration); *// returns 10,000,000 for a second.*

duration = duration;

}

}

}

return ready;

}

bool **Mp3**::Play()

{

if (ready&&pimc)

{

HRESULT hr = pimc->Run();

return SUCCEEDED(hr);

}

return false;

}

bool **Mp3**::Pause()

{

if (ready&&pimc)

{

HRESULT hr = pimc->Pause();

return SUCCEEDED(hr);

}

return false;

}

bool **Mp3**::Stop()

{

if (ready&&pimc)

{

HRESULT hr = pimc->Stop();

return SUCCEEDED(hr);

}

return false;

}

bool **Mp3**::WaitForCompletion(long msTimeout, long\* EvCode)

{

if (ready&&pimex)

{

HRESULT hr = pimex->WaitForCompletion(msTimeout, EvCode);

return \*EvCode > 0;

}

return false;

}

bool **Mp3**::SetVolume(long vol)

{

if (ready&&piba)

{

HRESULT hr = piba->put\_Volume(vol);

return SUCCEEDED(hr);

}

return false;

}

long **Mp3**::GetVolume()

{

if (ready&&piba)

{

long vol = -1;

HRESULT hr = piba->get\_Volume(&vol);

if(SUCCEEDED(hr))

return vol;

}

return -1;

}

\_\_int64 **Mp3**::GetDuration()

{

return duration;

}

\_\_int64 **Mp3**::GetCurrentPosition()

{

if (ready&&pims)

{

\_\_int64 curpos = -1;

HRESULT hr = pims->GetCurrentPosition(&curpos);

if(SUCCEEDED(hr))

return curpos;

}

return -1;

}

bool **Mp3**::SetPositions(\_\_int64\* pCurrent, \_\_int64\* pStop, bool bAbsolutePositioning)

{

if (ready&&pims)

{

DWORD flags = 0;

if(bAbsolutePositioning)

flags = AM\_SEEKING\_AbsolutePositioning | AM\_SEEKING\_SeekToKeyFrame;

else

flags = AM\_SEEKING\_RelativePositioning | AM\_SEEKING\_SeekToKeyFrame;

HRESULT hr = pims->SetPositions(pCurrent, flags, pStop, flags);

if(SUCCEEDED(hr))

return true;

}

return false;

}

#include "**Mp3**.h"

void main()

{

*// Initialize COM*

::CoInitialize(NULL);

std::wcout<<L"Enter the **MP3** path: ";

std::wstring path;

getline(wcin, path);

std::wcout<<path<<std::endl;

**Mp3** **mp3**;

int status = 0;

if(**mp3**.Load(path.c\_str()))

{

status = SV\_LOADED;

}

else *// Error*

{

*// ...*

}

if(**mp3**.Play())

{

status = SV\_PLAYING;

}

else *// Error*

{

*// ...*

}

*// ... after some time*

if(**mp3**.Stop())

{

status = SV\_STOPPED;

}

else *// Error*

{

*// ...*

}

*// Uninitialize COM*

::CoUninitialize();

}