# **[C++标准库实现WAV文件读写](https://www.cnblogs.com/wangguchangqing/p/5970516.html) <https://www.cnblogs.com/wangguchangqing/p/5970516.html>**

void CRecordDlg::OnFile()

{

// TODO: Add your control notification handler code here

CFileDialog dlg(FALSE,

"wav",

"Noname",

OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT,

"Wave File(\*.wav)|\*.wav||");

if(dlg.DoModal() == IDOK)

{

m\_strFile=dlg.GetPathName();

UpdateData(FALSE);

}

}

以上 CString m\_strFile;为CRecordDlg的成员函数

CScope m\_scope;

int m\_nBits; //采样值保存位数

int m\_nSpeed;//采样速度变量

int m\_nChannel; //声道变量

Long m\_lTime; //录音时间变量

CString m\_strFile; //编辑控件成员变量，指向带路径的文件名字符串。

//////////////////////////////////////////////////////////////////////////////////////////////////////

开始录音 按钮

void CRecordDlg::OnRecord()

{

// TODO: Add your control notification handler code here

UpdateData(TRUE);//将控件的值赋值给成员变量

if(!recording)//recording=0时允许录音，等待录音

{

PrepareFormat();//格式准备

CreateWaveFile((LPSTR)(LPCSTR)m\_strFile,&params);

RecordStart();//开始录音 //录音正常结束时recording=TRUE，否则recording=FALSE

m\_lTime=0;

UpdateData(FALSE);//将成员变量的值赋值给控件显示

if(recording)//录音正常结束时recording=TRUE

SetDlgItemText(IDC\_RECORDBT,"停止录音(&T)");//将开始录音按钮显示为停止录音

}

else //以前RecordStart()录音正常结束时recording=TRUE，已经录音，此时录音按钮为停止录音

{

RecordEnd();//结束录音的收尾工作，断开麦克风，释放内存

CloseWaveFile(&params);

if(!recording)

SetDlgItemText(IDC\_RECORDBT,"开始录音(&R)");

}

}

///////////////

void CRecordDlg::PrepareFormat()

{

long bps=1;

wfm.wFormatTag=WAVE\_FORMAT\_PCM;

wfm.cbSize=sizeof(WAVEFORMATEX);

if(m\_nChannel == 1)

{

wfm.nChannels=2;

bps\*=2;

}

else

wfm.nChannels=1;

if(m\_nBits == 1)

{

wfm.wBitsPerSample=16;

bps\*=2;

}

else

wfm.wBitsPerSample=8;

wfm.nBlockAlign=(unsigned short)bps;

if(m\_nSpeed == 2)

{

wfm.nSamplesPerSec=(DWORD)44100;

bps\*=44100;

}

else if(m\_nSpeed == 1)

{

wfm.nSamplesPerSec=(DWORD)22050;

bps\*=22050;

}

else

{

wfm.nSamplesPerSec=(DWORD)11025;

bps\*=11025;

}

wfm.nAvgBytesPerSec=bps;

}

//////////////////////

BOOL CreateWaveFile(LPSTR filename,WriteParams \*pParams)

{

MMRESULT result;

HMMIO hmmio;

pParams->mmciRiff=new MMCKINFO;

pParams->mmciFmt=new MMCKINFO;

pParams->mmciData=new MMCKINFO;

//Create waveform file

hmmio = mmioOpen(filename, NULL,

MMIO\_CREATE | MMIO\_WRITE | MMIO\_ALLOCBUF);

pParams->hmmio=hmmio;

//Create Global chunk

memset(pParams->mmciRiff,0,sizeof(MMCKINFO));

pParams->mmciRiff->fccType=mmioFOURCC('W', 'A', 'V', 'E');

result=mmioCreateChunk(hmmio,pParams->mmciRiff,MMIO\_CREATERIFF);

if(result != MMSYSERR\_NOERROR)

return FALSE;

//Create format chunk

memset(pParams->mmciFmt,0,sizeof(MMCKINFO));

pParams->mmciFmt->ckid=mmioFOURCC('f', 'm', 't', ' ');

result=mmioCreateChunk(hmmio,pParams->mmciFmt,0);

if(result != MMSYSERR\_NOERROR)

return FALSE;

//Write waveform format

mmioWrite(hmmio, (char\*)pParams->pwf,

sizeof(PCMWAVEFORMAT));

//Adjust format chunk size

mmioAscend(hmmio,pParams->mmciFmt,0);

//Create data chunk

memset(pParams->mmciData,0,sizeof(MMCKINFO));

pParams->mmciData->ckid=mmioFOURCC('d', 'a', 't', 'a');

result=mmioCreateChunk(hmmio,pParams->mmciData,0);

if(result != MMSYSERR\_NOERROR)

return FALSE;

char strFile[MAX\_PATH];

sprintf(strFile,"%s.dat",filename);

g\_fpWaveFile=fopen(strFile,"w");

return TRUE;

}

/////////////////

BOOL RecordStart()

{

int i;

if(hwi != NULL)

return FALSE;

//Buffer size

DWORD bufsize=BUF\_SIZE;

UINT uID;

MMRESULT result;

//Is there any device can use

uID=waveInGetNumDevs();

if(uID == 0)

goto ERROR\_END;

uID=WAVE\_MAPPER;

WAVEINCAPS wic;

//Get device caps

result=waveInGetDevCaps(uID,&wic,sizeof(WAVEINCAPS));

if(result != MMSYSERR\_NOERROR)

goto ERROR\_END;

//Set wave formats

wfm.nChannels=1;

wfm.nSamplesPerSec=(DWORD)22050;

wfm.wBitsPerSample=8;

wfm.wFormatTag=WAVE\_FORMAT\_PCM;

wfm.nBlockAlign=1;

wfm.nAvgBytesPerSec=(DWORD)22050;

wfm.cbSize=sizeof(WAVEFORMATEX);

//Open device

result=waveInOpen(&hwi,

uID,

&wfm,

(DWORD)waveInProc,

(DWORD)AfxGetInstanceHandle(),

CALLBACK\_FUNCTION);

if(result != MMSYSERR\_NOERROR)

goto ERROR\_END;

for(i=0;i<BUF\_NUMS;i++)//BUF\_NUMS=5

{

//Alloc buffers

memset(wh+i,0,sizeof(WAVEHDR));

wh[i].lpData=(char \*)GlobalAlloc(GMEM\_FIXED | GMEM\_SHARE,bufsize);

wh[i].dwBufferLength=bufsize;

//Prepare buffers

result=waveInPrepareHeader(hwi,wh+i,sizeof(WAVEHDR));

if(result != MMSYSERR\_NOERROR)

goto ERROR\_END;

//Add buffers

result=waveInAddBuffer(hwi,wh+i,sizeof(WAVEHDR));

if(result != MMSYSERR\_NOERROR)

goto ERROR\_END;

}

//Start recording

result=waveInStart(hwi);

if(result != MMSYSERR\_NOERROR)

goto ERROR\_END;

recording=1;

return TRUE;

ERROR\_END:

for(i=0;i<BUF\_NUMS;i++)

{

if(wh[i].lpData != NULL)

{

waveInUnprepareHeader(hwi,wh+i,sizeof(WAVEHDR));

GlobalFree(wh[i].lpData);

}

}

if(hwi != NULL)

{

waveInClose(hwi);

hwi=NULL;

}

return FALSE;

}

////////////////////

//End recording

void RecordEnd()

{

int i;

if(hwi == NULL)//hwi未关联录音设备

return;

MMRESULT result;

//Stop

result=waveInStop(hwi);//mmsystem中的函数，停止输入设备（麦克风），成功返回0

recording=0;//重新将录音功能设置为可录音，允许下一次录音

//Reset buffers

result=waveInReset(hwi);//waveInReset:mmsystem中的函数,重置语音输入，设备终止输入, 位置清0; 放弃未处理的缓冲区并返回给程序

//为下次录音准备

for(i=0;i<BUF\_NUMS;i++)

{

//Unprepare and release buffers

waveInUnprepareHeader(hwi,wh+i,sizeof(WAVEHDR));

GlobalFree(wh[i].lpData);

}

//Close device

result=waveInClose(hwi);//关闭指定的波形输入设备,成功返回 0

hwi=NULL;//取消hwi的指向录音设备，再录音时需重新指向

}

///////

BOOL CloseWaveFile(WriteParams \*pParams)

{

MMRESULT result;

if(pParams->hmmio == NULL)

return FALSE;

//Adjust data chunk

mmioAscend(pParams->hmmio,pParams->mmciData,0);

//Adjust RIFF chunk

mmioAscend(pParams->hmmio,pParams->mmciRiff,0);

//Close waveform file

result=mmioClose(pParams->hmmio,0);//mmsystem中文件，关闭用mmioOpen打开的设备文件.

if(result != MMSYSERR\_NOERROR)

return FALSE;

pParams->hmmio=NULL;//解除指向

delete pParams->mmciRiff;//释放内存

delete pParams->mmciFmt;//释放内存

delete pParams->mmciData;//释放内存

if(g\_fpWaveFile != NULL)

{

fclose(g\_fpWaveFile);//关闭在CreateWaveFile函数中打开的文件 g\_fpWaveFile=fopen(strFile,"w");

g\_fpWaveFile=NULL;//解除指向

}

return TRUE;

}

/////////////////////////////////////////////////////////////////////////////////////////////////////

播放语音

void CRecordDlg::OnPlay()

{

// TODO: Add your control notification handler code here

LPSTR pData;

LONG len;

UpdateData(TRUE);

PlayStart(m\_strFile,&wfm,&pData,&len);

}

BOOL PlayStart(LPCSTR file,WAVEFORMATEX \*pwf,LPSTR \*pData,LONG \*pLen)

{

HMMIO hmmio=NULL;

MMCKINFO ckiRiff,cki;

MMRESULT result;

LONG len;

if(hwo != NULL) //回放设备已打开

{

result=waveOutReset(hwo);//停止放音

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

result=waveOutClose(hwo);//关闭打开的回放设备

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

hwo=NULL;//关闭打开的回放设备之后让设备回到空闲备用

}

hmmio=mmioOpen((LPSTR)file,NULL,MMIO\_READ | MMIO\_ALLOCBUF);//为输入输出打开一个多媒体文件file

if(hmmio == NULL)

return FALSE;

ckiRiff.fccType=mmioFOURCC('W','A','V','E');

result=mmioDescend(hmmio,&ckiRiff,NULL,MMIO\_FINDRIFF);

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

cki.ckid=mmioFOURCC('f','m','t',' ');

result=mmioDescend(hmmio,&cki,&ckiRiff,MMIO\_FINDCHUNK);

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

len=mmioRead(hmmio,(LPSTR)pwf,sizeof(WAVEFORMATEX));//hmmio：文件句柄,被读取的文件的句柄。pwf：指向一个缓冲区,包含/存放从文件读取的数据。sizeof(WAVEFORMATEX)要从文件读取的字节数

if(len == -1)

goto END\_ERROR;

result=mmioAscend(hmmio,&cki,0);

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

cki.ckid=mmioFOURCC('d','a','t','a');

result=mmioDescend(hmmio,&cki,&ckiRiff,MMIO\_FINDCHUNK);

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

\*pLen=cki.cksize;

\*pData=(LPSTR)GlobalAlloc(GMEM\_FIXED | GMEM\_SHARE,\*pLen); //分配内存地址数\*pLen所指向的大小的内存，GMEM\_FIXED分配固定的内存,返回值是一个指针

//若函数调用成功,则返回一个新分配的内存对象的句柄,如果堆内没有足够的空间满足请求,函数将返回 NULL

if(\*pData == NULL)

goto END\_ERROR;

len=mmioRead(hmmio,(LPSTR)\*pData,\*pLen);//!!从打开的文件hmmio中读取长度为\*pLen的数据到pData所指向的地址中

if(len == -1)

goto END\_ERROR;

result=waveOutOpen(&hwo,//一个指向接收波形音频输出装置柄的缓冲器，返回设备句柄的指针

WAVE\_MAPPER,//将要被打开的波形音频输出装置的ID,指定为WAVE\_MAPPER时函数会根据给定的波形格式选择合适的设备

pwf, //波形结构的指针，包含要申请的波形格式

NULL,//回调函数地址或窗口指针，不使用回调函数则用NULL

(DWORD)AfxGetInstanceHandle(),//传递到CALLBACK进程的用户实例数据。如果是窗口CALLBACK进程的话，该参数不用

CALLBACK\_NULL// 默认的设置，即无CALLBACK进程

);//这个函数打开一个给定的波形音频输出装置来进行回放

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

memset(&wh2,0,sizeof(WAVEHDR)); //WAVEHDR wh2; wh2为全局变量

/\*typedef struct {

LPSTR lpData; //波形缓冲数据(传入首地址)

DWORD dwBufferLength; //缓冲区长度

DWORD dwBytesRecorded; //指明录音时缓冲区容量

DWORD dwUser; //用户数据

DWORD dwFlags; //提供缓冲区标示

DWORD dwLoops; //循环次数

struct wavehdr\_tag \* lpNext; //预留,NULL

DWORD reserved; //预留,0

} WAVEHDR;\*/

wh2.lpData=\*pData;

wh2.dwBufferLength=\*pLen;

result=waveOutPrepareHeader(hwo,&wh2,sizeof(WAVEHDR));//准备一个波形数据块用于播放

if(result != MMSYSERR\_NOERROR)

goto END\_ERROR;

//::MessageBox(NULL, "msg1", "caption", MB\_OK);

waveOutWrite(hwo,&wh2,sizeof(WAVEHDR));//播放语音,mmsystem函数,将一个数据块发送到一个指定的波形音频输出装置。在播放语音的同时运行后面的程序

//hwo:HWAVEOUT波形音频输出装置的柄（handle）,&wh2: 一个指向包含有数据块信息的WAVEHDR结构的指针

//第三个参数就是WAVEHDR结构的大小（用sizeof (WAVEHDR) 就可以了）

//::MessageBox(NULL, "msg2", "caption", MB\_OK);

return TRUE;

END\_ERROR:

if(\*pData != NULL)

{

GlobalFree(\*pData);

\*pData=NULL;

\*pLen=0;

}

if(hmmio != NULL)

mmioClose(hmmio,0);

if(hwo != NULL)

waveOutClose(hwo);

}