Windows录音API学习笔记

结构体和函数信息

结构体

**WAVEINCAPS**

该结构描述了一个波形音频输入设备的能力。

typedef struct {

    WORD      wMid; 用于波形音频输入设备的设备驱动程序制造商标识符。

    WORD      wPid; 声音输入设备的产品识别码。

    MMVERSION vDriverVersion; 用于波形音频输入设备的设备驱动程序的版本号。高位字节是主版本号，低字节是次版本号。

    CHAR      szPname[MAXPNAMELEN]; 设备名称

    DWORD     dwFormats; 所支持的标准格式。可以是以下组合：

    WORD      wChannels; 数值指定设备是否支持单（1）或立体声（2）的输入

    WORD      wReserved1; 填充

} WAVEINCAPS;

HWAVEIN 目前推测是打开声音设备后获取的句柄（现在看就是句柄）

**WAVEHDR**

此结构定义用于标识一个波形音频缓冲器中的报头。

typedef struct {

    LPSTR  lpData; 指向波形缓冲区。

    DWORD  dwBufferLength; 缓冲区长度

    DWORD  dwBytesRecorded; 被用于输入时缓冲区的数据长度

    DWORD  dwUser; 用户数据

    DWORD  dwFlags; 标志提供有关缓冲区的信息。详见MSDN

    DWORD  dwLoops; 循环播放的次数，仅用于输出缓冲器

    struct wavehdr\_tag \* lpNext; 保留

    DWORD  reserved; 保留

} WAVEHDR;

**WAVEFORMATEX**

该结构定义的波形的音频数据的格式。只有共同所有波形的音频数据格式的格式信息被包括在这种结构。对于需要更多的信息格式，该结构被包括在另一种结构第一部件，伴随着的附加信息。

typedef struct {

    WORD  wFormatTag; 波形音频格式类型。格式标记注册的微软公司的很多压缩算法。格式标签的完整列表可以在MMREG.H头文件中找到。

    WORD  nChannels; 在波形的音频数据的通道数。单声道的数据使用一个通道，立体声数据使用两个通道。

    DWORD nSamplesPerSec; 采样速率，以每秒（赫兹）的样品，每个通道应被播放或录制。如果wFormatTag是WAVE\_FORMAT\_PCM，然后nSamplesPerSec共同的价值观是8.0千赫，11.025千赫，22.05 kHz和44.1 kHz的。对于非PCM格式，这件必须根据制造商的格式标记的规范来计算。

    DWORD nAvgBytesPerSec; 所需的平均数据传输速率，以每秒字节的格式标记。如果wFormatTag是WAVE\_FORMAT\_PCM，nAvgBytesPerSec应等于nSamplesPerSec和nBlockAlign的乘积。对于非PCM格式，这件必须根据制造商的格式标记的规范来计算。

播放和录制软件，可以通过使用nAvgBytesPerSec成员估计缓冲区大小。

WORD  nBlockAlign;

块对齐，以字节为单位。块对齐用于wFormatTag格式类型数据的最小基本单位。如果wFormatTag是WAVE\_FORMAT\_PCM，nBlockAlign应等于nChannels和wBitsPerSample的乘积/8（每字节比特）。对于非PCM格式，这件必须根据制造商的格式标记的规范来计算。

播放和录制软件必须处理的数据nBlockAlign字节的倍数的时间。书面和从设备读取数据必须开始于一个块的开始。例如，它是非法的开始播放在样品的中间PCM数据（也就是，在一个非块对齐的边界）。

WORD  wBitsPerSample;

采样位数用于wFormatTag格式类型。如果wFormatTag是WAVE\_FORMAT\_PCM，然后wBitsPerSample应等于8或16。对于非PCM格式，这件必须根据制造商的格式标记的规范来设置。请注意，某些压缩方案不能定义为wBitsPerSample一个值，因此该构件可以是零。

WORD  cbSize;

大小以字节为单位的额外格式信息追加到WAVEFORMATEX结构的末端。此信息可用于通过非PCM格式来存储额外用于wFormatTag属性。如果没有额外的信息所必需的wFormatTag，这个部件必须被设置为零。请注意，对于WAVE\_FORMAT\_PCM格式（只有WAVE\_FORMAT\_PCM格式），这个成员被忽略。

} WAVEFORMATEX;

**函数信息**

//这里获取到声音设备句柄使用的是waveInOpen函数，在那里进行的初始化。

返回装置的数量。返回的零值表示没有设备存在或发生了错误。

UINT **waveInGetNumDevs**(VOID);

该函数检索一个给定的波形音频输入设备的能力。

MMRESULT **waveInGetDevCaps**(

UINT*uDeviceID*,     标识符波形音频输出的装置。它可以是一个设备标识符或一个开放波形音频输入设备的一个句柄。

  LPWAVEINCAPS*pwic*,  指向一个WAVEINCAPS结构填充有关器件的信息的功能

  UINT*cbwic*WAVEINCAPS结构体的字节数

);

MMRESULT **waveInOpen**(

  LPHWAVEIN*phwi*,指向打开的声音设备的标识句柄。在fdwOpen参数指定为WAVE\_FORMAT\_QUERY时这个参数可以为空

  UINT*uDeviceID*, 要打开设备的标识符

  LPWAVEFORMATEX*pwfx*, 指向一个WAVEFORMATEX结构，它标识用于记录波形音频数据所需的格式。您可以waveInOpen返回后立即释放此结构。

  DWORD*dwCallback*, 指针指向一个固定的回调函数，事件句柄，句柄到窗口或线程波形音频录制过程中被调用来处理与记录的进度消息的标识符。如果没有回调功能是必需的，则该值可以是零。

  DWORD*dwCallbackInstance*,传给回调机制的数据类型，此参数并不用于窗口回调

  DWORD*fdwOpenz*指明dwCallBack传入的数据类型，比如时间句柄或者函数指针

);

该函数为音频输入设备提供缓冲区

MMRESULT **waveInPrepareHeader**(

  HWAVEIN*hwi*, 声音输入设备句柄

  LPWAVEHDR*pwh*, 指向WAVEHDR结构，标识要准备的缓冲区。

  UINT*cbwh*WAVEHDR结构的大小

);

该函数发送一个输入缓冲区给定的波形音频输入设备。当缓冲区被填满后，通知应用程序。

MMRESULT **waveInAddBuffer**(

  HWAVEIN*hwi*, 声音输入设备句柄

  LPWAVEHDR*pwh*, 指向WAVEHDR结构，标识要准备的缓冲区。

  UINT*cbwh*WAVEHDR结构的大小

);

该函数启动指定设备的输入

MMRESULT **waveInStart**(

  HWAVEIN*hwi* 设备句柄

);

基于Windows API的录音分析

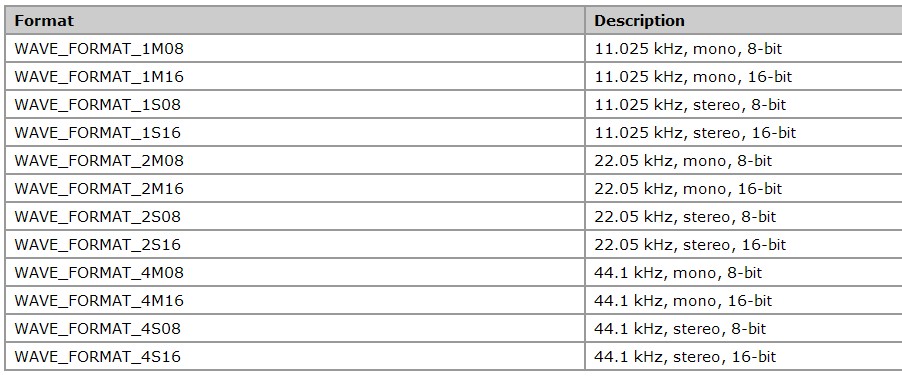
   昨天看了一天的Audio API和windows的Wav文件相关资料，渐渐的理清了一点思路，所以在此总结一下，待本文已完成之后，应该就能继续下一步了。

    1 首先要了解的是计算机是如何表示声音文件的。

    我们熟知的音频格式有：MP3，WMA，FLAC，以及WAV。这里我暂时只关注WAV。要知道的是，WAV其实就是WAVE，意思为波形。真实世界中的声音都是连续的，因为是模拟信号，但是在计算机中存储的信息都是数字信号。所以在将声音存储到计算机之前，就必须要进行声音的数字化，转换成计算机能够存储的形式。

    学过信号与系统的应该都知道，模拟信号转换为数字信号，一种比较通用的方法就是进行等间隔采样。根据奈奎斯特定理，采样频率至少为信号频率的2倍，才能无失真的保存原有的音频信号。因此采样频率的高低决定了数字信号的保真度，自然是越高越好。打个比方，一个周期为1ms的正弦信号，采两个点和采100个点的信号在还原成模拟信号的时候，肯定是采100个点信号的还原效果更好。

     在对模拟信号完成采样后，得到的是一系列的离散电压信号。因此在将数据存储至计算机前，还需要对这些模拟信号进行量化。这里所谓量化，就是用二进制数据来表示电平的大小。一般采用8位（256级）或者16位（65536级）的数据来表示，在硬件级的设计中，需要根据ADC的具体情况来决定。而在Windows中，可以使用waveInGetDevCaps函数获取声卡信息，以判断使用8位或者16位的量化采样位数。在其参数的**dwFormats** 成员中，包含了相应的信息，具体如下：



    假设0-5V电平的正弦信号，在采用8位的量化下，就将0-5V分成256个梯度，每个梯度的电压差值为0.0195V。一个4V的采样值对应的数值就是205(204.8)。

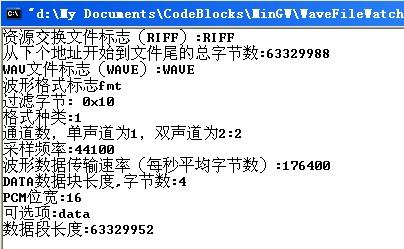
在经过上面两步的处理之后，模拟声音信号就转化为了具有固定采样频率和相应量化标准的数字信号。所以数字声音信号最主要的两个参数就是采样位宽和采样频率。

另外一个需要注意的就是声道。我们平时听到的音乐都是立体声，也就是双声道的。这个在WAV文件格式中有专门的介绍，不过我没细看……记得是交替出现的。

现在回过头来看一下上面的表，声卡在提供信号的时候，也就上面这三个主要因素：采样频率，声道，量化位宽。

    我根据WAV文件头的资料写了一份代码，打开了并查看了一个我计算机上的WAV格式的音乐文件，以前在CD上拷的。

    运行结果如下：



    不过这个就是看一看，暂时还用不到，有了上面的基础，接下来要做的事分析API的使用。

第一个要调用的函数：

UINT **waveInGetNumDevs**(VOID);

    这个函数的仅仅是用来查看计算机上是否有声卡设备，如果返回值是0，那么表明没有声卡。不过一般这种情况不会发生，毕竟现在的计算机主板都是集成声卡的。

    第二个要调用的函数：

            MMRESULT **waveInGetDevCaps**(

　　　　　　　　　　 UINT*uDeviceID*,

    　　　　　　　　  LPWAVEINCAPS*pwic*,

                           UINT*cbwic*);

    这个函数的第一个参数是设备ID，不过我们现在并不知道设备ID，但是没关系，只要我们知道有设备存在就可以了。不过在MSDN上有这么一句让我比较费解的话：

“Use this function to determine the number of waveform-audio input devices present in the system.”

中文意思就是：使用这个函数来决定系统中声音输入设备的数量。我看这个函数名感觉更像是获取设备能力信息，而且第二个参数更是直接点出了其目的，“声音输入设备的能力”。

回来接着说第一个参数，MSDN在Remarks说明了这个参数可以是从0值到设备数量中的任何一个数，或者使用WAVE\_MAPPER。我估计这个WAVE\_MAPPER就是0值，因为传入的参数要求是个指针，所以，你懂的。因此只要填个0就可以了。不过至于多声卡设备具体怎么玩，我还真的不太清楚。

第二个参数是一个指向WAVEINCAPS结构的指针，在这把这个结构好好剖析一下：

typedef struct {

    WORD      wMid;

    WORD      wPid;

    MMVERSION vDriverVersion;

    CHAR      szPname[MAXPNAMELEN];

    DWORD     dwFormats;

    WORD      wChannels;

    WORD      wReserved1;

} WAVEINCAPS;

前两个结构体成员分别是设备制造商信息和产品标示符，我点击去看了下，发现一堆的设备制造商信息的宏定义……不用太关心。

第三个成员是MMVERSION，其本质就是一个UINT，驱动设备版本号，也不用太关心。

第四个成员是设备名称，就是一个以’\0’结尾的字符串，同样不用管太多。

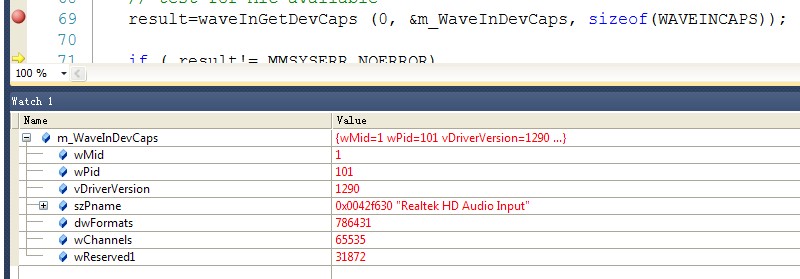
第五个成员很重要，因为指明了支持音频信号标准的组合，具体看这个结构体的MSDN翻译。

第六个是声道支持的判断。其实这个参数在第五个参数的分析中就能看出来的。

第七个是保留参数，不过至今还没用上。

第三个参数仅仅是告知第二个参数对应结构体的大小。

根据程序的调试结果看，这个函数的用途更多是初始化WAVEINCAPS。根据下面的图可以看到，在该函数调用完毕后，结构体被初始化。



第三个调用的函数：

**MMRESULT waveInOpen(**

**LPHWAVEIN***phwi***,**

**UINT***uDeviceID***,**

**LPWAVEFORMATEX***pwfx***,**

**DWORD***dwCallback***,**

**DWORD***dwCallbackInstance***,**

**DWORD***fdwOpen*

**);**

    这个函数很有意思，同时也相对比较复杂。这里的第一个参数是设备句柄，不过在这里的这个参数是在这里被初始化的，传进来就可以了。第二个参数还是设备ID，不过这个设备ID好像也是只传个0就行了。第三个参数需要好好研究一下，也就是这个结构，**WAVEFORMATEX**。我现在觉得是用来初始化设备的。

先查看一下MSDN中对其的说明：

typedef struct {

    WORD  wFormatTag;

    WORD  nChannels;

    DWORD nSamplesPerSec;

    DWORD nAvgBytesPerSec;

    WORD  nBlockAlign;

    WORD  wBitsPerSample;

    WORD  cbSize;

} WAVEFORMATEX;

第一个成员是格式类型参数，在第二个函数中，也就是waveInGetDevCaps函数的第二个参数的子成员Format，其组成基本是一致的。如果在这个参数中使用上面的宏定义指定，那么后面的如nChannel参数函数不用处理了。不过一种比较通用的方式还是将wFormatTag修改为WAVE\_FORMAT\_PCM，并依此初始化后面的参数。

这第三个参数可以根据需求进行相应的初始化，比如这形式的：

WaveInitFormat(

WORD    nCh,

DWORD   nSampleRate,

WORD    BitsPerSample)

{

      m\_WaveFormat.wFormatTag = WAVE\_FORMAT\_PCM;

      m\_WaveFormat.nChannels = nCh;

      m\_WaveFormat.nSamplesPerSec = nSampleRate;

      m\_WaveFormat.nAvgBytesPerSec = nSampleRate \* nCh \* BitsPerSample/8;

      m\_WaveFormat.nBlockAlign = m\_WaveFormat.nChannels \* BitsPerSample/8;

      m\_WaveFormat.wBitsPerSample = BitsPerSample;

      m\_WaveFormat.cbSize = 0;

}

在这个初始化函数中，第一个是WAVE\_FORMAT\_PCM，第二个就是声道数的选项了，第三个是采样频率，第四个参数是根据采样频率，声道，每个采样点字节数的乘积计算的。第五个参数则是块对齐大小，也就是每个独立的声音节点信息的大小，其计算公式就是声道数\*每个采样点的字节数。第六个参数就是采样点的位数，指定了声音的量化水平。第七个参数有点说法，不过在wFormatTag被指定为WAVE\_FORMAT\_PCM的时候，这个参数会被忽略。

第四参数是回调处理参数，比如，一个回调函数，或者事件句柄，或者线程句柄。用于通知应用程序处理音频数据的一个参数。等操作系统的音频信号接收完毕，就是通过这个参数来决定以什么方式来把数据传递给应用程序。

第五个参数是传给回调机制的数据类型，并不用语窗口回调。这个参数我不太懂，但是使用的时候，传入NULL就可以了。

第六个参数是打开设备的一个通知信息，主要用途就是告诉函数上面第四个参数是一个回调函数还是一个事件句柄。就这么简单。

第四个调用的函数是waveInPrepareHeader。这个函数的功能就是在初始化好后，为声音输入设备准备一块存放声音数据的缓冲区。

MMRESULT waveInPrepareHeader(

  HWAVEIN*hwi*,

  LPWAVEHDR*pwh*,

  UINT*cbwh*

);

第一个参数就是设备句柄，不多说。

第二个参数需要好好的关注一下，这一个结构体。

typedef struct {

    LPSTR  lpData;

    DWORD  dwBufferLength;

    DWORD  dwBytesRecorded;

    DWORD  dwUser;

    DWORD  dwFlags;

    DWORD  dwLoops;

    struct wavehdr\_tag \* lpNext;

    DWORD  reserved;

} WAVEHDR;

         LpData就是一个缓冲区，指向你所提供的内存空间。至于具体大小，好像没有什么限制，因为填满了会通知你，到时候用专门的函数去处理就OK了。

       DwBufferLength参数就是缓冲区长度，不多说。

       DwByteRecorded实际记录的字节数，因为未必每次都会用掉全部的缓冲区。

       DwUser用户数据，具体干啥，MSDN没说，有可能是到时候传给回调函数的信息，我猜的。

       DwFlags对于buffer的附加信息，根据MSDN的要求，在使用这个函数waveInPrepareHeader的时候，这个参数必须为0。

       DwLoops循环播放的次数，只用于输出使用。这里用不到。

       LpNext 这个虽然是指针，但是实际确实一个保留参数。

       reserved最后一个和上面一样。

    这样这个结构体算是完事了。就是提供一个头部输入的缓冲区。

    第三个参数就是上面那个结构的字节数。一个sizeof万事大吉。

    这个函数的用途就是为声音输入设备准备一个缓冲区，到时候音频信息就会被写进这个内存区域内。

    第五个函数就是waveInAddBuffer，这个函数的用途就是把函数waveInPrepareHeader准备好的那个缓冲区发送给声音输入设备。

MMRESULT waveInAddBuffer(

  HWAVEIN*hwi*,

  LPWAVEHDR*pwh*,

  UINT*cbwh*

);

    和上一个函数的原型是一致的，三个参数和waveInPrepareHeader函数使用过的参数必须一致。而且在本函数调用前，*pwh*参数的Buffer必须经过waveInPrepareHeader函数的处理。当缓冲区被填满，那么会通知应用程序。详见MSDN。

    第六个函数是waveInStart，这个函数仅用于开启录音功能，最简单。

    Windows的录音流程大致如下：

    1 先查看本地机器是否拥有声音输入设备。

    2 获取声音输入设备的信息

    一般上面的两部不是很必要，毕竟现在的电脑基本都拥有集成声卡。不过从稳定性和通用性看，还是很必要的。

    3 打开设备，获取设备句柄，传入对应的事件句柄。

4 准备一个异步线程专门用于录音完成后的处理工作，并等待事件。

    5 通过设备句柄为其准备缓冲区

    6 将准备好的缓冲区通过句柄添加到设备中

    接下来属于系统的工作，正常情况下，在缓冲区被填满后，将会触发事件，来通知异步线程进行处理。获取声音信息后，要再次添加缓冲区，才能继续录音。

20140118 补充：

　　有时在缓冲区的建立上，一般会采取栈分配或者堆分配的方式。栈分配内存的析构处理通过退栈完成，用户不用手动处理。但是堆分配的时候就会遇到麻烦，具体如下：

　　在准备缓冲区的时候会调用waveInPrepareHeader函数，这个函数调用后，为其分配的内存就无法通过delete或者free来释放了，因为在该函数调用后这块内存区域被锁定了。此时必须调用waveInUnprepareHeader函数才能解锁定，然后才能释放。

　　但是在调用waveInPrepareHeader函数后，再接着调用了waveInAddBuffer函数，在该缓冲区未被填满的时候，尝试使用waveInUnprepareHeader函数解锁定，就会返回失败码33。这里的解决方案就是在决定释放空间前，首先调用一个函数：waveInReset。这个函数调用后，就可以将内存从waveInAddBuffer函数的限定中释放，然后再常规使用waveInUnprepareHeader释放，最后调用delete或者free释放内存空间。