**WAVE文件格式**

一、综述  
    WAVE文件作为多媒体中使用的声波文件格式之一，它是以RIFF格式为标准的。  
RIFF是英文Resource Interchange File Format的缩写，每个WAVE文件的头四个  
字节便是“RIFF”。  
    WAVE文件是由若干个Chunk组成的。按照在文件中的出现位置包括：RIFF WAVE  
Chunk, Format Chunk, Fact Chunk(可选), Data Chunk。具体见下图：

------------------------------------------------  
|             RIFF WAVE Chunk                  |  
|             ID  = 'RIFF'                     |  
|             RiffType = 'WAVE'                |  
------------------------------------------------  
|             Format Chunk                     |  
|             ID = 'fmt '                      |  
------------------------------------------------  
|             Fact Chunk(optional)             |  
|             ID = 'fact'                      |  
------------------------------------------------  
|             Data Chunk                       |  
|             ID = 'data'                      |  
------------------------------------------------  
            图1   Wav格式包含Chunk示例

    其中除了Fact Chunk外，其他三个Chunk是必须的。每个Chunk有各自的ID，位  
于Chunk最开始位置，作为标示，而且均为4个字节。并且紧跟在ID后面的是Chunk大  
小（去除ID和Size所占的字节数后剩下的其他字节数目），4个字节表示，低字节  
表示数值低位，高字节表示数值高位。下面具体介绍各个Chunk内容。  
PS：  
    所有数值表示均为低字节表示低位，高字节表示高位。

二、具体介绍  
RIFF WAVE Chunk  
    ==================================  
    |       |所占字节数|  具体内容   |  
    ==================================  
    | ID    |  4 Bytes |   'RIFF'    |  
    ----------------------------------  
    | Size  |  4 Bytes |             |  
    ----------------------------------  
    | Type  |  4 Bytes |   'WAVE'    |  
    ----------------------------------  
            图2  RIFF WAVE Chunk

    以'FIFF'作为标示，然后紧跟着为size字段，该size是整个wav文件大小减去ID  
和Size所占用的字节数，即FileLen - 8 = Size。然后是Type字段，为'WAVE'，表  
示是wav文件。  
    结构定义如下：  
 struct RIFF\_HEADER  
 {  
  char szRiffID[4];  // 'R','I','F','F'  
  DWORD dwRiffSize;  
  char szRiffFormat[4]; // 'W','A','V','E'  
 };

Format Chunk  
    ====================================================================  
    |               |   字节数  |              具体内容                |  
    ====================================================================  
    | ID            |  4 Bytes  |   'fmt '                             |  
    --------------------------------------------------------------------  
    | Size          |  4 Bytes  | 数值为16或18，18则最后又附加信息     |  
    --------------------------------------------------------------------  ----  
    | FormatTag     |  2 Bytes  | 编码方式，一般为0x0001               |     |  
    --------------------------------------------------------------------     |  
    | Channels      |  2 Bytes  | 声道数目，1--单声道；2--双声道       |     |  
    --------------------------------------------------------------------     |  
    | SamplesPerSec |  4 Bytes  | 采样频率                             |     |  
    --------------------------------------------------------------------     |  
    | AvgBytesPerSec|  4 Bytes  | 每秒所需字节数                       |     |===> WAVE\_FORMAT  
    --------------------------------------------------------------------     |  
    | BlockAlign    |  2 Bytes  | 数据块对齐单位(每个采样需要的字节数) |     |  
    --------------------------------------------------------------------     |  
    | BitsPerSample |  2 Bytes  | 每个采样需要的bit数                  |     |  
    --------------------------------------------------------------------     |  
    |               |  2 Bytes  | 附加信息（可选，通过Size来判断有无） |     |  
    --------------------------------------------------------------------  ----  
                            图3  Format Chunk

    以'fmt '作为标示。一般情况下Size为16，此时最后附加信息没有；如果为18  
则最后多了2个字节的附加信息。主要由一些软件制成的wav格式中含有该2个字节的  
附加信息。  
    结构定义如下：  
 struct WAVE\_FORMAT  
 {  
  WORD wFormatTag;  
  WORD wChannels;  
  DWORD dwSamplesPerSec;  
  DWORD dwAvgBytesPerSec;  
  WORD wBlockAlign;  
  WORD wBitsPerSample;  
 };  
 struct FMT\_BLOCK  
 {  
  char  szFmtID[4]; // 'f','m','t',' '  
  DWORD  dwFmtSize;  
  WAVE\_FORMAT wavFormat;  
 };

Fact Chunk  
    ==================================  
    |       |所占字节数|  具体内容   |  
    ==================================  
    | ID    |  4 Bytes |   'fact'    |  
    ----------------------------------  
    | Size  |  4 Bytes |   数值为4   |  
    ----------------------------------  
    | data  |  4 Bytes |             |  
    ----------------------------------  
            图4  Fact Chunk

    Fact Chunk是可选字段，一般当wav文件由某些软件转化而成，则包含该Chunk。  
    结构定义如下：  
 struct FACT\_BLOCK  
 {  
  char  szFactID[4]; // 'f','a','c','t'  
  DWORD  dwFactSize;  
 };

Data Chunk  
    ==================================  
    |       |所占字节数|  具体内容   |  
    ==================================  
    | ID    |  4 Bytes |   'data'    |  
    ----------------------------------  
    | Size  |  4 Bytes |             |  
    ----------------------------------  
    | data  |          |             |  
    ----------------------------------  
             图5 Data Chunk

    Data Chunk是真正保存wav数据的地方，以'data'作为该Chunk的标示。然后是  
数据的大小。紧接着就是wav数据。根据Format Chunk中的声道数以及采样bit数，  
wav数据的bit位置可以分成以下几种形式：  
    ---------------------------------------------------------------------  
    |   单声道  |    取样1    |    取样2    |    取样3    |    取样4    |  
    |           |--------------------------------------------------------  
    |  8bit量化 |    声道0    |    声道0    |    声道0    |    声道0    |  
    ---------------------------------------------------------------------  
    |   双声道  |          取样1            |           取样2           |  
    |           |--------------------------------------------------------  
    |  8bit量化 |  声道0(左)  |  声道1(右)  |  声道0(左)  |  声道1(右)  |  
    ---------------------------------------------------------------------  
    |           |          取样1            |           取样2           |  
    |   单声道  |--------------------------------------------------------  
    | 16bit量化 |    声道0    |  声道0      |    声道0    |  声道0      |  
    |           | (低位字节)  | (高位字节)  | (低位字节)  | (高位字节)  |  
    ---------------------------------------------------------------------  
    |           |                         取样1                         |  
    |   双声道  |--------------------------------------------------------  
    | 16bit量化 |  声道0(左)  |  声道0(左)  |  声道1(右)  |  声道1(右)  |  
    |           | (低位字节)  | (高位字节)  | (低位字节)  | (高位字节)  |  
    ---------------------------------------------------------------------  
                         图6 wav数据bit位置安排方式

    Data Chunk头结构定义如下：  
    struct DATA\_BLOCK  
 {  
  char szDataID[4]; // 'd','a','t','a'  
  DWORD dwDataSize;  
 };

 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

WAVE文件格式说明表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 偏移地址 | 字节数 | 数据类型 | 内   容 |
| 文件头 | 00H | 4 | char | "RIFF"标志 |
| 04H | 4 | long int | 文件长度 |
| 08H | 4 | char | "WAVE"标志 |
| 0CH | 4 | char | "fmt"标志 |
| 10H | 4 |  | 过渡字节（不定） |
| 14H | 2 | int | 格式类别（10H为PCM形式的声音数据) |
| 16H | 2 | int | 通道数，单声道为1，双声道为2 |
| 18H | 4 | int | 采样率（每秒样本数），表示每个通道的播放速度， |
| 1CH | 4 | long int | 波形音频数据传送速率，其值为通道数×每秒数据位数×每样本的数据位数／8。播放软件利用此值可以估计缓冲区的大小。 |
| 20H | 2 | int | 数据块的调整数（按字节算的），其值为通道数×每样本的数据位值／8。播放软件需要一次处理多个该值大小的字节数据，以便将其值用于缓冲区的调整。 |
| 22H | 2 |  | 每样本的数据位数，表示每个声道中各个样本的数据位数。如果有多个声道，对每个声道而言，样本大小都一样。 |
| 24H | 4 | char | 数据标记符＂data＂ |
| 28H | 4 | long int | 语音数据的长度 |

　　PCM数据的存放方式：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 样本1 | | 样本2 | |
| 8位单声道 | 0声道 | | 0声道 | |
| 8位立体声 | 0声道（左） | 1声道（右） | 0声道（左） | 1声道（右） |
| 16位单声道 | 0声道低字节 | 0声道高字节 | 0声道低字节 | 0声道高字节 |
| 16位立体声 | 0声道（左）低字节 | 0声道（左）高字节 | 1声道（右）低字节 | 1声道（右）高字节 |

　WAVE文件的每个样本值包含在一个整数i中，i的长度为容纳指定样本长度所需的最小字节数。首先存储低有效字节，表示样本幅度的位放在i的高有效位上，剩下的位置为0，这样8位和16位的PCM波形样本的数据格式如下所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 样本大小 | 数据格式 | 最大值 | 最小值 |
| 8位PCM | unsigned int | 225 | 0 |
| 16位PCM | int | 32767 | -32767 |

文件头长度加起来是44个字节（用UltraEdit打开一个WAVE文件，数一下就知道了）。如果用以个结构体来定义WAVE文件头应该为：  
struct WAVEFILEHEADER  
{  
 char chRIFF[4];  
 DWORD dwRIFFLen;  
 char chWAVE[4];  
 char chFMT[4];  
 DWORD dwFMTLen;  
 PCMWAVEFORMAT pwf;  
 char chDATA[4];  
 DWORD dwDATALen;  
};