目录

WinAPI: waveInAddBuffer - 向波形输入设备发送一个输入缓冲区

提示:

缓冲区写满后送回应用程序.

在缓冲区给 waveInAddBuffer 前, 先要调用 waveInPrepareHeader 准备; 还要调用 GlobalAlloc 给 TWaveHdr 和其中 lpData 指向的缓冲区分配内存(使用 GMEM_MOVEABLE、GMEM_SHARE), 并用 GlobalLock 锁定.

```
//声明:
waveInAddBuffer(
  hWaveIn: HWAVEIN;         {波形输入设备句柄}
  lpWaveInHdr: PWaveHdr; {TWaveHdr 结构的指针}
  uSize: UINT                {TWaveHdr 结构大小}
): MMRESULT;                  {成功返回 0; 可能的错误值如下:}

MMSYSERR_INVALHANDLE = 5;   {设备句柄无效}
WAVERR_UNPREPARED      = 34; {没准备好缓冲区}
MMSYSERR_HANDLEBUSY   = 12; {设备已被另一线程使用}

//TWaveHdr 是 wavehdr_tag 结构的重定义
wavehdr_tag = record
  lpData: PChar;              {指向波形数据缓冲区}
  dwBufferLength: DWORD;   {波形数据缓冲区的长度}
  dwBytesRecorded: DWORD; {若首部用于输入，指出缓冲区中的数据量}
  dwUser: DWORD;              {指定用户的 32 位数据}
  dwFlags: DWORD;            {缓冲区标志}
  dwLoops: DWORD;            {循环播放次数，仅用于输出缓冲区}
  lpNext: PWaveHdr;         {保留}
  reserved: DWORD;          {保留}
end;
```

```
//dwFlags 的可选值:
WHDR_DONE         = $00000001; {设备已使用完缓冲区，并返回给程序}
WHDR_PREPARED     = $00000002; {waveInPrepareHeader 或
waveOutPrepareHeader 已将缓冲区准备好}
WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}
WHDR_ENDLOOP    = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}
WHDR_INQUEUE    = $00000010; { reserved for driver }
```

WinAPI: waveInGetPosition - 获取当前输入设备的输入位置

```
//声明:
waveInGetPosition(
  hWaveIn: HWAVEIN; {设备句柄}
  lpInfo: PMMTime;  {TMMTime 结构的指针}
  uSize: UINT         {TMMTime 结构大小}
): MMRESULT;          {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}

//TMMTime 是 mmtime_tag 结构的重定义:
mmtime_tag = record
```

```
  case wType: UINT of
   TIME_MS:       (ms: DWORD);            {毫米}
   TIME_SAMPLES:(sample: DWORD);       {波形音频取样数}
   TIME_BYTES:   (cb: DWORD);            {波形音频字节数(字节偏移量)}
   TIME_TICKS:   (ticks: DWORD);        {TICK 数}
   TIME_SMPTE:(                          {动画及电视协会的 SMPTE 时间，是个内嵌结
构}
      hour: Byte;                   {时}
      min: Byte;                    {分}
      sec: Byte;                    {秒}
      frame: Byte;                  {帧}
      fps: Byte;                    {每秒帧数}
      dummy: Byte;                  {填充字节(为对齐而用)}
      pad: array[0..1] of Byte); {}
    TIME_MIDI : (songptrpos: DWORD); {MIDI 时间}
end;


//使用 TMMTime 结构前，应先指定 TMMTime.wType :
TIME_MS         = $0001; {默认；打开或复位时将回到此状态}
TIME_SAMPLES = $0002;
TIME_BYTES     = $0004;
TIME_SMPTE     = $0008;
```

```
TIME_MIDI      = $0010;

TIME_TICKS     = $0020;
```

WinAPI: waveInGetNumDevs - 获取波形输入设备的数目

```
//声明:

waveInGetNumDevs: UINT; {无参数; 返回波形输入设备的数目}
```

WinAPI: waveInGetID - 获取输入设备 ID

```
//声明:

waveInGetID(

  hWaveIn: HWAVEIN;  {获取输入设备句柄}

  lpuDeviceID: PUINT {接受 ID 的变量的指针}

): MMRESULT;         {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}

MMSYSERR_HANDLEBUSY  = 12; {设备已被另一线程使用}
```

WinAPI: waveInClose - 关闭指定的波形输入设备

提示：若 waveInAddBuffer 送出的缓冲区未返回则失败；可用 waveInReset 放弃所有未用完的缓冲区.

```
//声明:
waveInClose(
  hWaveIn: HWAVEIN {设备句柄；函数若成功返回，句柄则不再有效}
): MMRESULT;        {成功返回 0；可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
WAVERR_STILLPLAYING  = 33; {缓冲区还在队列中}
```

WinAPI: waveInGetDevCaps - 查询输入设备的性能

```
//声明:
waveInGetDevCaps(
  hwo: HWAVEOUT;          {输入设备 ID; HWAVEIN ?}
  lpCaps: PWaveInCaps; {TWaveInCaps 结构的指针，用于接受设备信息}
  uSize: UINT             {TWaveInCaps 结构大小}
): MMRESULT;              {成功返回 0；可能的错误值见下:}
```

```
MMSYSERR_BADDEVICEID = 2; {设备 ID 超界}

MMSYSERR_NODRIVER      = 6; {没有安装驱动程序}


//TWaveInCaps 是 tagWAVEINCAPSA 结构的重定义:

tagWAVEINCAPSA = record

  wMid: Word;                                          {制造商 ID}

  wPid: Word;                                          {产品 ID}

  vDriverVersion: MMVERSION;                           {版本号; 高字节是主版本号, 低字节是次版本号}

  szPname: array[0..MAXPNAMELEN-1] of AnsiChar; {产品名称}

  dwFormats: DWORD;                                    {支持的格式}

  wChannels: Word;                                     {单声道(1)还是立体声(2)}

  wReserved1: Word;                                    { structure packing }

end;


//dwFormats:

WAVE_INVALIDFORMAT = $00000000; {invalid format}

WAVE_FORMAT_1M08     = $00000001; {11.025 kHz, Mono,    8-bit }

WAVE_FORMAT_1S08     = $00000002; {11.025 kHz, Stereo, 8-bit }

WAVE_FORMAT_1M16     = $00000004; {11.025 kHz, Mono,    16-bit}

WAVE_FORMAT_1S16     = $00000008; {11.025 kHz, Stereo, 16-bit}
```

```
WAVE_FORMAT_2M08    = $00000010; {22.05   kHz, Mono,    8-bit }
WAVE_FORMAT_2S08    = $00000020; {22.05   kHz, Stereo, 8-bit }
WAVE_FORMAT_2M16    = $00000040; {22.05   kHz, Mono,    16-bit}
WAVE_FORMAT_2S16    = $00000080; {22.05   kHz, Stereo, 16-bit}
WAVE_FORMAT_4M08    = $00000100; {44.1    kHz, Mono,    8-bit }
WAVE_FORMAT_4S08    = $00000200; {44.1    kHz, Stereo, 8-bit }
WAVE_FORMAT_4M16    = $00000400; {44.1    kHz, Mono,    16-bit}
WAVE_FORMAT_4S16    = $00000800; {44.1    kHz, Stereo, 16-bit}
```

WinAPI: waveInGetErrorText - 根据错误号得到错误描述

提示: 错误文本的长度一般不超过 MAXERRORLENGTH = 128; 如果缓冲区太小, 文本会被截断.

```
-------------------------------------------------------------------
------------
//声明:
waveInGetErrorText(
  mmrError: MMRESULT; {错误号}
  lpText: PChar;      {缓冲区}
  uSize: UINT         {缓冲区大小}
): MMRESULT;          {成功返回 0; 失败再返回错误号, 可能的错误是:}
```

```
MMSYSERR_BADERRNUM = 9; {错误号超界}
```

WinAPI: waveInMessage - 向波形输入设备发送一条消息

```
//声明:
waveInMessage(
  hWaveIn: HWAVEIN; {设备句柄}
  uMessage: UINT;    {消息}
  dw1: DWORD          {消息参数}
  dw2: DWORD          {消息参数}
): MMRESULT;          {将由设备给返回值}
```

WinAPI: waveInOpen - 打开波形输入设备

提示: 因为其中的回调函数是在中断时间内访问的, 必须在 DLL 中; 要访问的数据都必须是在固定的数据段中; 除了

PostMessage

timeGetSystemTime

timeGetTime

timeSetEvent

timeKillEvent

midiOutShortMsg

midiOutLongMsg

OutputDebugString 外, 也不能有其他系统调用.

//声明:

```
waveInOpen(
    lphWaveIn: PHWAVEIN;          {用于返回设备句柄的指针; 如果 dwFlags=WAVE_FORMAT_QUERY, 这里应是 nil}
  uDeviceID: UINT;            {设备 ID; 可以指定为: WAVE_MAPPER, 这样函数会根据给定的波形格式选择合适的设备}
  lpFormatEx: PWaveFormatEx; {TWaveFormat 结构的指针; TWaveFormat 包含要申请的波形格式}
  dwCallback: DWORD           {回调函数地址或窗口句柄; 若不使用回调机制, 设为 nil}
  dwInstance: DWORD           {给回调函数的实例数据; 不用于窗口}
  dwFlags: DWORD              {打开选项}
): MMRESULT;                  {成功返回 0; 可能的错误值见下:}


MMSYSERR_BADDEVICEID = 2;  {设备 ID 超界}
MMSYSERR_ALLOCATED    = 4;  {指定的资源已被分配}
MMSYSERR_NODRIVER     = 6;  {没有安装驱动程序}
MMSYSERR_NOMEM        = 7;  {不能分配或锁定内存}
```

```
WAVERR_BADFORMAT       = 32; {设备不支持请求的波形格式}


//TWaveFormatEx 结构:
TWaveFormatEx = packed record
  wFormatTag: Word;         {指定格式类型; 默认 WAVE_FORMAT_PCM = 1;}
  nChannels: Word;          {指出波形数据的声道数; 单声道为 1, 立体声为 2}
  nSamplesPerSec: DWORD;  {指定采样频率(每秒的样本数)}
  nAvgBytesPerSec: DWORD; {指定数据传输的传输速率(每秒的字节数)}
  nBlockAlign: Word;        {指定块对齐(每个样本的字节数), 块对齐是数据的最小单位}
  wBitsPerSample: Word;    {采样大小(字节), 每个样本的量化位数}
  cbSize: Word;             {附加信息的字节大小}
end;
{16 位立体声 PCM 的块对齐是 4 字节(每个样本 2 字节, 2 个通道)}


//打开选项 dwFlags 的可选值:
WAVE_FORMAT_QUERY = $0001;      {只是判断设备是否支持给定的格式, 并不打开}
WAVE_ALLOWSYNC     = $0002;      {当是同步设备时必须指定}
CALLBACK_WINDOW    = $00010000; {当 dwCallback 是窗口句柄时指定}
CALLBACK_FUNCTION = $00030000; {当 dwCallback 是函数指针时指定}


//如果选择窗口接受回调信息, 可能会发送到窗口的消息有:
MM_WIM_OPEN   = $3BE;
```

MM_WIM_CLOSE = $3BF;

MM_WIM_DATA = $3C0;

//如果选择函数接受回调信息, 可能会发送给函数的消息有:

WIM_OPEN = MM_WIM_OPEN;

WIM_CLOSE = MM_WIM_CLOSE;

WIM_DATA = MM_WIM_DATA;

WinAPI: waveInPrepareHeader - 为波形输入准备一个缓冲区

提示: 必须调用 GlobalAlloc 给 TWaveHdr 和其中的 lpData 指向的缓冲区分配内存(使用 GMEM_MOVEABLE、GMEM_SHARE), 并用 GlobalLock 锁定.

```
//声明:

waveInPrepareHeader(

  hWaveIn: HWAVEIN;       {设备句柄}

  lpWaveInHdr: PWaveHdr; {TWaveHdr 结构的指针}

  uSize: UINT             {TWaveHdr 结构大小}

): MMRESULT;               {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
```

```
MMSYSERR_NOMEM          = 7;  {不能分配或锁定内存}

MMSYSERR_HANDLEBUSY   = 12; {其他线程正在使用该设备}


//TWaveHdr 是 wavehdr_tag 结构的重定义

wavehdr_tag = record

  lpData: PChar;             {指向波形数据缓冲区}

  dwBufferLength: DWORD;  {波形数据缓冲区的长度}

  dwBytesRecorded: DWORD; {若 TWaveHdr 用于输入，指出缓冲区中的数据量}

  dwUser: DWORD;             {指定用户的 32 位数据}

  dwFlags: DWORD;            {缓冲区标志}

  dwLoops: DWORD;            {循环播放次数，仅用于输出缓冲区}

  lpNext: PWaveHdr;         {保留}

  reserved: DWORD;          {保留}

end;


//TWaveHdr 中的 dwFlags 的可选值:

WHDR_DONE        = $00000001; {设备已使用完缓冲区，并返回给程序}

WHDR_PREPARED             =       $00000002;       {waveInPrepareHeader       或

waveOutPrepareHeader 已将缓冲区准备好}

WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}

WHDR_ENDLOOP    = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}

WHDR_INQUEUE     = $00000010; { reserved for driver }
```

WinAPI: waveInReset - 重置输入

提示:

函数会终止输入，位置清 0；放弃未处理的缓冲区并返回给程序；

TWaveHdr 结构中的 dwBytesRecorded 将包含实际数据的长度.

```
//声明:
waveInReset(
  hWaveIn: HWAVEIN {设备句柄}
): MMRESULT;        {成功返回 0；可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY  = 12; {设备已被另一线程使用}
```

WinAPI: waveInStart - 启动输入

```
//声明:
waveInStart(
  hWaveIn: HWAVEIN {设备句柄}
): MMRESULT;        {成功返回 0；可能的错误值见下:}
```

MMSYSERR_INVALHANDLE = 5; {设备句柄无效}

MMSYSERR_HANDLEBUSY = 12; {设备已被另一线程使用}

WinAPI: waveInStop - 停止输入

提示: 如果未启动则调用无效, 但也返回 0; 缓冲区会被返回, TWaveHdr 结构中的 dwBytesRecorded 将包含返回的实际数据的长度.

//声明:

waveInStop(

  hWaveIn: HWAVEIN {设备句柄}

): MMRESULT;   {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5; {设备句柄无效}

MMSYSERR_HANDLEBUSY = 12; {设备已被另一线程使用}

WinAPI: waveInUnprepareHeader - 清除由 waveInPrepareHeader 完成的准备

提示:

设备写满缓冲区返回给程序后，须调用此函数;

释放(GlobalFree)缓冲区前，须调用此函数;

取消一个尚未准备的缓冲区将无效，但函数返回 0

```
------------------------------------------------------------------------------

//声明:
waveInUnprepareHeader(
  hWaveIn: HWAVEIN;         {设备句柄}
  lpWaveInHdr: PWaveHdr; {TWaveHdr 结构的指针}
  uSize: UINT                 {TWaveHdr 结构大小}
): MMRESULT;                 {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY  = 12; {设备已被另一线程使用}
WAVERR_STILLPLAYING  = 33; {缓冲区还在队列中}

//TWaveHdr 是 wavehdr_tag 结构的重定义
wavehdr_tag = record
```

```
  lpData: PChar;            {指向波形数据缓冲区}
  dwBufferLength: DWORD;  {波形数据缓冲区的长度}
  dwBytesRecorded: DWORD; {若首部用于输入，指出缓冲区中的数据量}
  dwUser: DWORD;            {指定用户的 32 位数据}
  dwFlags: DWORD;           {缓冲区标志}
  dwLoops: DWORD;           {循环播放次数，仅用于输出缓冲区}
  lpNext: PWaveHdr;         {保留}
  reserved: DWORD;          {保留}
end;
```

```
//TWaveHdr 中的 dwFlags 的可选值:
WHDR_DONE      = $00000001; {设备已使用完缓冲区，并返回给程序}
WHDR_PREPARED  = $00000002; {waveInPrepareHeader 或 waveOutPrepareHeader 已将缓冲区准备好}
WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}
WHDR_ENDLOOP   = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}
WHDR_INQUEUE   = $00000010; { reserved for driver }
```

WinAPI: waveOutBreakLoop - 跳出循环

提示:

循环是由 saveOutWrite 传递的 TWaveHdr 结构的 dwLoop 和 dwFlags 控制的;

dwFlags 的 WHDR_BEGINLOOP、WHDR_ENDLOOP 标识循环的开始和结束数据块;

在同一数据块上循环，应同时指定这两个标志;

循环次数 dwLoops 应该在开始块上指定;

循环终止前，组成循环体的块一定要播放完;

当无播放内容或循环设定失败时，函数也能返回 0.

```
//声明:
waveOutBreakLoop(
  hWaveOut: HWAVEOUT {设备句柄}
): MMRESULT;          {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY   = 12; {设备已被另一线程使用}



//TWaveHdr 是 wavehdr_tag 结构的重定义
wavehdr_tag = record
  lpData: PChar;            {指向波形数据缓冲区}
  dwBufferLength: DWORD;  {波形数据缓冲区的长度}
  dwBytesRecorded: DWORD; {若首部用于输入，指出缓冲区中的数据量}
  dwUser: DWORD;            {指定用户的 32 位数据}
  dwFlags: DWORD;            {缓冲区标志}
  dwLoops: DWORD;            {循环播放次数，仅用于输出缓冲区}
  lpNext: PWaveHdr;          {保留}
  reserved: DWORD;           {保留}
end;
```

```
//TWaveHdr 中的 dwFlags 的可选值:
WHDR_DONE      = $00000001; {设备已使用完缓冲区，并返回给程序}
WHDR_PREPARED  = $00000002; {waveInPrepareHeader 或 waveOutPrepareHeader 已将缓冲区准备好}
WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}
WHDR_ENDLOOP   = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}
WHDR_INQUEUE   = $00000010; {保留(给设备)}
```

WinAPI: waveOutClose - 关闭设备

提示: 若正在播放，应先调用 waveOutReset 终止播放，然后再关闭，不然会失败.

```
//声明:
waveOutClose(
  hWaveOut: HWAVEOUT {设备句柄}
): MMRESULT;         {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}

MMSYSERR_HANDLEBUSY  = 12; {设备已被另一线程使用}

WAVERR_STILLPLAYING  = 33; {缓冲区还在队列中}
```

WinAPI: waveOutGetDevCaps - 查询输出设备的性能

```
//声明:

waveOutGetDevCaps(

  uDeviceID: UINT;          {输出设备 ID}

  lpCaps: PWaveOutCaps;     {TWaveOutCaps 结构的指针, 用于接受设备信息}

  uSize: UINT): MMRESULT; {TWaveOutCaps 结构大小}

): MMRESULT;                {成功返回 0; 可能的错误值见下:}


MMSYSERR_BADDEVICEID = 2; {设备 ID 超界}

MMSYSERR_NODRIVER    = 6; {没有安装驱动程序}


//TWaveOutCaps 是 tagWAVEOUTCAPSA 结构的重定义:

tagWAVEOUTCAPSA = record

  wMid: Word;                                   {制造商 ID}

  wPid: Word;                                   {产品 ID}
  vDriverVersion: MMVERSION;                    {版本号；高字节是主版本号，低字节是次版本号}
  szPname: array[0..MAXPNAMELEN-1] of AnsiChar; {产品名称}
  dwFormats: DWORD;                             {支持的格式}
  wChannels: Word;                              {单声道(1)还是立体声(2)}
  dwSupport: DWORD;                             {其他功能}
end;


//dwFormats:
WAVE_INVALIDFORMAT = $00000000; {invalid format}
WAVE_FORMAT_1M08   = $00000001; {11.025 kHz, Mono,   8-bit }
WAVE_FORMAT_1S08   = $00000002; {11.025 kHz, Stereo, 8-bit }
WAVE_FORMAT_1M16   = $00000004; {11.025 kHz, Mono,   16-bit}
WAVE_FORMAT_1S16   = $00000008; {11.025 kHz, Stereo, 16-bit}
WAVE_FORMAT_2M08   = $00000010; {22.05  kHz, Mono,   8-bit }
WAVE_FORMAT_2S08   = $00000020; {22.05  kHz, Stereo, 8-bit }
WAVE_FORMAT_2M16   = $00000040; {22.05  kHz, Mono,   16-bit}
WAVE_FORMAT_2S16   = $00000080; {22.05  kHz, Stereo, 16-bit}
WAVE_FORMAT_4M08   = $00000100; {44.1   kHz, Mono,   8-bit }
WAVE_FORMAT_4S08   = $00000200; {44.1   kHz, Stereo, 8-bit }
WAVE_FORMAT_4M16   = $00000400; {44.1   kHz, Mono,   16-bit}
WAVE_FORMAT_4S16   = $00000800; {44.1   kHz, Stereo, 16-bit}
```

```
//dwSupport:
WAVECAPS_PITCH           = $0001; {支持音调控制}
WAVECAPS_PLAYBACKRATE    = $0002; {支持播放速度控制}
WAVECAPS_VOLUME          = $0004; {支持音量控制}
WAVECAPS_LRVOLUME        = $0008; {支持左右声道音量控制}
WAVECAPS_SYNC            = $0010; {}
WAVECAPS_SAMPLEACCURATE = $0020; {}
WAVECAPS_DIRECTSOUND     = $0040; {}
```

WinAPI: waveOutGetID - 获取输出设备 ID

```
//声明:
waveOutGetID(
  hWaveOut: HWAVEOUT; {设备句柄}
  lpuDeviceID: PUINT  {接受 ID 的变量的指针}
): MMRESULT;          {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY  = 12; {设备已被另一线程使用}
```

WinAPI: waveOutGetNumDevs - 获取波形输出设备的数目

```
//声明:
waveOutGetNumDevs: UINT; {无参数; 返回波形输出设备的数目}
```

WinAPI: waveOutGetPitch - 获取输出设备当前的音调设置(音高的倍数值)

提示:

参数 lpdwPitch 虽然指向的是 4 字节的正整数, 但表示的是个小数;

两个高位表示整数部分, 两个低位表示小数部分;

$8000 表示一半, $4000 表示四分之一;

譬如: $00010000 表示 1.0, 说明音高没变; $000F8000, 表示 15.5 倍;

修改音高不会改变播放速度、采样速度和播放时间, 但不是所有设备都支持.

```
//声明:
waveOutGetPitch(
  hWaveOut: HWAVEOUT; {设备句柄}
  lpdwPitch: PDWORD   {存放音高值的变量的指针}
): MMRESULT;          {成功返回 0; 可能的错误值见下:}
```

```
MMSYSERR_INVALHANDLE   = 5;  {设备句柄无效}

MMSYSERR_NOTSUPPORTED = 8;  {设备不支持}

MMSYSERR_HANDLEBUSY    = 12; {设备已被另一线程使用}
```

WinAPI: waveOutGetPlaybackRate - 获取输出设备当前的播放速度设置(默认速度值的倍数)

提示:

参数 lpdwRate 虽然指向的是 4 字节的正整数, 但表示的是个小数;

 两个高位表示整数部分, 两个低位表示小数部分;

 $8000 表示一半, $4000 表示四分之一;

 譬如: $00010000 表示 1.0, 说明速度没有改变变; $000F8000, 表示 15.5 倍;

 修改播放速度不会改变采样速度, 但肯定会改变播放时间.

```
----------------------------------------------------------------------
------------

//声明:

waveOutGetPlaybackRate(

  hWaveOut: HWAVEOUT; {设备句柄}

  lpdwRate: PDWORD     {存放速度值的变量的指针}

): MMRESULT;             {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE   = 5;  {设备句柄无效}
```

```
MMSYSERR_NOTSUPPORTED = 8;  {设备不支持}

MMSYSERR_HANDLEBUSY    = 12; {设备已被另一线程使用}
```

WinAPI: waveOutGetPosition - 获取输出设备当前的播放位置

```
//声明:

waveOutGetPosition(

  hWaveOut: HWAVEOUT; {设备句柄}

  lpInfo: PMMTime;     {TMMTime 结构的指针, 用于返回播放位置}

  uSize: UINT            {TMMTime 结构的大小, 以字节为单位}

): MMRESULT;              {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE  = 5;  {设备句柄无效}

MMSYSERR_HANDLEBUSY    = 12; {设备已被另一线程使用}


//TMMTime 是 mmtime_tag 结构的重定义:

mmtime_tag = record

  case wType: UINT of

    TIME_MS:       (ms: DWORD);            {毫米}

    TIME_SAMPLES:(sample: DWORD);       {波形音频取样数}

    TIME_BYTES:   (cb: DWORD);            {波形音频字节数(字节偏移量)}

    TIME_TICKS:   (ticks: DWORD);        {TICK 数}
```

```
   TIME_SMPTE:(                    {动画及电视协会的SMPTE时间，是个内嵌结构}
      hour: Byte;                {时}
      min: Byte;                 {分}
      sec: Byte;                 {秒}
      frame: Byte;               {帧}
      fps: Byte;                 {每秒帧数}
      dummy: Byte;                 {填充字节(为对齐而用)}
      pad: array[0..1] of Byte); {}
    TIME_MIDI : (songptrpos: DWORD); {MIDI时间}
end;

//使用TMMTime结构前，应先指定TMMTime.wType :
TIME_MS      = $0001; {默认；打开或复位时将回到此状态}
TIME_SAMPLES = $0002;
TIME_BYTES   = $0004;
TIME_SMPTE   = $0008;
TIME_MIDI    = $0010;
TIME_TICKS   = $0020;
```

WinAPI: waveOutGetVolume - 获取输出设备当前的音量设置

提示:

参数 lpdwVolume 的两低位字节存放左声道音量，两高位字节存放右声道音量;

$FFFF、$0000 分别表示最大与最小音量;

如不支持立体声，两低位字节存放单声道音量.

```
//声明:
waveOutGetVolume(
  hwo: HWAVEOUT;       {设备句柄}
  lpdwVolume: PDWORD {存放音量值的变量的指针}
): MMRESULT;           {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE   = 5;  {设备句柄无效}
MMSYSERR_NODRIVER      = 6;  {没有安装驱动程序}
MMSYSERR_NOTSUPPORTED = 8;  {设备不支持}
```

WinAPI: waveOutMessage - 向波形输出设备发送一条消息

```
//声明:
```

```
waveOutMessage(
  hWaveOut: HWAVEOUT; {设备句柄}
  uMessage: UINT;       {消息}
  dw1: DWORD            {消息参数}
  dw2: DWORD            {消息参数}
): Longint;             {将由设备给返回值}
```

WinAPI: waveOutOpen - 打开波形输出设备

提示: 因为其中的回调函数是在中断时间内访问的, 必须在 DLL 中; 要访问的数据都必须是在固定的数据段中; 除了

PostMessage

timeGetSystemTime

timeGetTime

timeSetEvent

timeKillEvent

midiOutShortMsg

midiOutLongMsg

OutputDebugString 外, 也不能有其他系统调用.

//声明:

```
waveOutOpen(
    lphWaveOut: PHWaveOut;      {用于返回设备句柄的指针; 如果 dwFlags=WAVE_FORMAT_QUERY, 这里应是 nil}
  uDeviceID: UINT;          {设备 ID; 可以指定为: WAVE_MAPPER, 这样函数会根据给定的波形格式选择合适的设备}
  lpFormat: PWaveFormatEx; {TWaveFormat 结构的指针; TWaveFormat 包含要申请的波形格式}
  dwCallback: DWORD          {回调函数地址或窗口句柄; 若不使用回调机制, 设为 nil}
  dwInstance: DWORD          {给回调函数的实例数据; 不用于窗口}
  dwFlags: DWORD             {打开选项}
): MMRESULT;                 {成功返回 0; 可能的错误值见下:}

MMSYSERR_BADDEVICEID = 2;  {设备 ID 超界}
MMSYSERR_ALLOCATED   = 4;  {指定的资源已被分配}
MMSYSERR_NODRIVER    = 6;  {没有安装驱动程序}
MMSYSERR_NOMEM       = 7;  {不能分配或锁定内存}
WAVERR_BADFORMAT     = 32; {设备不支持请求的波形格式}

//TWaveFormatEx 结构:
TWaveFormatEx = packed record
  wFormatTag: Word;        {指定格式类型; 默认 WAVE_FORMAT_PCM = 1;}
  nChannels: Word;          {指出波形数据的通道数; 单声道为 1, 立体声为 2}

  nSamplesPerSec: DWORD;  {指定样本速率(每秒的样本数)}

  nAvgBytesPerSec: DWORD; {指定数据传输的平均速率(每秒的字节数)}

  nBlockAlign: Word;        {指定块对齐(单位字节), 块对齐是数据的最小单位}

  wBitsPerSample: Word;    {采样大小(字节)}

  cbSize: Word;              {附加信息大小; PCM 格式没这个字段}

end;

{16 位立体声 PCM 的块对齐是 4 字节(每个样本 2 字节, 2 个通道)}
```

```
//打开选项 dwFlags 的可选值:

WAVE_FORMAT_QUERY = $0001;      {只是判断设备是否支持给定的格式, 并不打开}

WAVE_ALLOWSYNC      = $0002;      {当是同步设备时必须指定}

CALLBACK_WINDOW    = $00010000; {当 dwCallback 是窗口句柄时指定}

CALLBACK_FUNCTION = $00030000; {当 dwCallback 是函数指针时指定}
```

```
//如果选择窗口接受回调信息, 可能会发送到窗口的消息有:

MM_WOM_OPEN   = $3BB;

MM_WOM_CLOSE = $3BC;

MM_WOM_DONE   = $3BD;
```

```
//如果选择函数接受回调信息, 可能会发送给函数的消息有:

WOM_OPEN   = MM_WOM_OPEN;
```

```
WOM_CLOSE = MM_WOM_CLOSE;

WOM_DONE   = MM_WOM_DONE;
```

WinAPI: waveOutPause - 暂停播放

提示: 暂停后会保存当前位置, 可以用 waveOutRestart 从当前位置恢复播放. //声明:

```
waveOutPause(
  hWaveOut: HWAVEOUT {设备句柄}
): MMRESULT;           {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE   = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY    = 12; {设备已被另一线程使用}
```

WinAPI: waveOutPrepareHeader - 准备一个波形数据块用于播放

提示: 必须调用 GlobalAlloc 给 TWaveHdr 和其中的 lpData 指向的缓冲区分配内存 (使用 GMEM_MOVEABLE、GMEM_SHARE), 并用 GlobalLock 锁定.

//声明:

```
waveOutPrepareHeader(
  hWaveOut: HWAVEOUT;       {设备句柄}
  lpWaveOutHdr: PWaveHdr; {TWaveHdr 结构的指针}
  uSize: UINT                  {TWaveHdr 结构大小}
): MMRESULT;                   {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_NOMEM          = 7;  {不能分配或锁定内存}
MMSYSERR_HANDLEBUSY   = 12; {其他线程正在使用该设备}

//TWaveHdr 是 wavehdr_tag 结构的重定义
wavehdr_tag = record
  lpData: PChar;              {指向波形数据缓冲区}
  dwBufferLength: DWORD;  {波形数据缓冲区的长度}
  dwBytesRecorded: DWORD; {若首部用于输入，指出缓冲区中的数据量}
  dwUser: DWORD;              {指定用户的 32 位数据}
  dwFlags: DWORD;             {缓冲区标志}
  dwLoops: DWORD;             {循环播放次数，仅用于输出缓冲区}
  lpNext: PWaveHdr;          {保留}
  reserved: DWORD;           {保留}
end;
```

//TWaveHdr 中的 dwFlags 的可选值:

WHDR_DONE = $00000001; {设备已使用完缓冲区，并返回给程序}

WHDR_PREPARED = $00000002; {waveInPrepareHeader 或 waveOutPrepareHeader 已将缓冲区准备好}

WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}

WHDR_ENDLOOP = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}

WHDR_INQUEUE = $00000010; { reserved for driver }

WinAPI: waveOutReset - 重置输出

提示: 函数会终止输入，位置清 0; 放弃未处理的缓冲区并返回给程序.

//声明:

waveOutReset(

  hWaveOut: HWAVEOUT {设备句柄}

): MMRESULT; {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5; {设备句柄无效}

MMSYSERR_HANDLEBUSY = 12; {设备已被另一线程使用}

WinAPI: waveOutRestart - 重新启动一个被暂停的输出设备

提示: 当输出设备未暂停时调用该函数无效, 但也返回 0

```
----------------------------------------------------------------------
------------
//声明:
waveOutRestart(
  hWaveOut: HWAVEOUT {设备句柄}
): MMRESULT;           {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY  = 12; {设备已被另一线程使用}
```

WinAPI: waveOutSetPitch - 设置输出设备的音调设置(音高的倍数值)

提示:

参数 dwPitch 虽然是 4 字节的正整数, 但表示的是个小数;

 两个高位表示整数部分, 两个低位表示小数部分;

 $8000 表示一半, $4000 表示四分之一;

 譬如: $00010000 表示 1.0, 说明音高没变; $000F8000, 表示 15.5 倍;

 修改音高不会改变播放速度、采样速度和播放时间, 但不是所有设备都支持.

```
//声明:
waveOutSetPitch(
  hWaveOut: HWAVEOUT; {设备句柄}
  dwPitch: DWORD    {存放音高值的变量}
): MMRESULT;            {成功返回 0; 可能的错误值见下:}


MMSYSERR_INVALHANDLE   = 5;  {设备句柄无效}
MMSYSERR_NOTSUPPORTED = 8;  {设备不支持}
MMSYSERR_HANDLEBUSY    = 12; {设备已被另一线程使用}
```

WinAPI: waveOutGetPlaybackRate - 设置输出设备的播放速度(默认速度值的倍数)

提示:

参数 dwRate 虽然是 4 字节的正整数, 但表示的是个小数;

两个高位表示整数部分, 两个低位表示小数部分;

$8000 表示一半, $4000 表示四分之一;

譬如: $00010000 表示 1.0, 说明速度没有改变变; $000F8000, 表示 15.5 倍;

修改播放速度不会改变采样速度, 但肯定会改变播放时间.

```
//声明:
waveOutSetPlaybackRate(
  hWaveOut: HWAVEOUT; {设备句柄}
  dwRate: DWORD       {存放速度值的变量}
): MMRESULT;          {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE   = 5;  {设备句柄无效}
MMSYSERR_NOTSUPPORTED = 8;  {设备不支持}
MMSYSERR_HANDLEBUSY    = 12; {设备已被另一线程使用}
```

WinAPI: waveOutUnprepareHeader - 清除由 waveOutPrepareHeader 完成的准备

提示:

设备使用完数据块后，须调用此函数;

释放(GlobalFree)缓冲区前，须调用此函数;

取消一个尚未准备的缓冲区将无效，但函数返回 0

```
//声明:
waveOutUnprepareHeader(
  hWaveOut: HWAVEOUT;       {设备句柄}
  lpWaveOutHdr: PWaveHdr; {TWaveHdr 结构的指针}
  uSize: UINT                 {TWaveHdr 结构大小}
): MMRESULT;                  {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
MMSYSERR_HANDLEBUSY   = 12; {设备已被另一线程使用}
WAVERR_STILLPLAYING   = 33; {缓冲区还在队列中}

//TWaveHdr 是 wavehdr_tag 结构的重定义
wavehdr_tag = record
  lpData: PChar;              {指向波形数据缓冲区}
  dwBufferLength: DWORD;  {波形数据缓冲区的长度}
  dwBytesRecorded: DWORD; {若首部用于输入，指出缓冲区中的数据量}
  dwUser: DWORD;              {指定用户的 32 位数据}
  dwFlags: DWORD;             {缓冲区标志}
  dwLoops: DWORD;             {循环播放次数，仅用于输出缓冲区}
  lpNext: PWaveHdr;          {保留}
  reserved: DWORD;           {保留}
end;

//TWaveHdr 中的 dwFlags 的可选值:
```

WHDR_DONE = $00000001; {设备已使用完缓冲区，并返回给程序}

WHDR_PREPARED = $00000002; {waveInPrepareHeader 或 waveOutPrepareHeader 已将缓冲区准备好}

WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}

WHDR_ENDLOOP = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}

WHDR_INQUEUE = $00000010; { reserved for driver }

WinAPI: waveOutWrite - 向输出设备发送一个数据块

提示：把数据缓冲区传给 waveOutWrite 之前，必须使用 waveOutPrepareHeader 准备该缓冲区；

若未调用 waveOutPause 函数暂停设备，则第一次把数据块发送给设备时即开始播放.

```
//声明:
waveOutWrite(
  hWaveOut: HWAVEOUT;      {设备句柄}
  lpWaveOutHdr: PWaveHdr; {TWaveHdr 结构的指针}
  uSize: UINT              {TWaveHdr 结构大小}
): MMRESULT;               {成功返回 0; 可能的错误值见下:}

MMSYSERR_INVALHANDLE = 5;  {设备句柄无效}
```

```
MMSYSERR_HANDLEBUSY   = 12; {设备已被另一线程使用}

WAVERR_UNPREPARED      = 34; {未准备数据块}


//TWaveHdr 是 wavehdr_tag 结构的重定义

wavehdr_tag = record

  lpData: PChar;             {指向波形数据缓冲区}

  dwBufferLength: DWORD;  {波形数据缓冲区的长度}

  dwBytesRecorded: DWORD; {若首部用于输入，指出缓冲区中的数据量}

  dwUser: DWORD;             {指定用户的 32 位数据}

  dwFlags: DWORD;            {缓冲区标志}

  dwLoops: DWORD;            {循环播放次数，仅用于输出缓冲区}

  lpNext: PWaveHdr;         {保留}

  reserved: DWORD;          {保留}

end;


//TWaveHdr 中的 dwFlags 的可选值:

WHDR_DONE        = $00000001; {设备已使用完缓冲区，并返回给程序}

WHDR_PREPARED          =      $00000002;      {waveInPrepareHeader      或

waveOutPrepareHeader 已将缓冲区准备好}

WHDR_BEGINLOOP = $00000004; {缓冲区是循环中的第一个缓冲区，仅用于输出}

WHDR_ENDLOOP    = $00000008; {缓冲区是循环中的最后一个缓冲区，仅用于输出}

WHDR_INQUEUE     = $00000010; { reserved for driver }
```

合并两个 Wav 文件的函数

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.dfm}
```

{合并两个 Wav 文件的函数}

```
function ConWavFile(AWavFile1, AWavFile2, ANewFile: string): Boolean;
type
  TWavFormat = packed record
    ChunkID: array[0..3] of AnsiChar;     {'RIFF'}
    ChunkSize: Longword;                  {size-8}
    Format: array[0..3] of AnsiChar;      {'WAVE'}
    SubChunk1ID: array[0..3] of AnsiChar; {'fmt '}
    SubChunk1Size: Longword;              {hex10}
    AudioFormat: Word;                    {hex 01}
    NumOfChannels: Word;                  {1 mono, 2 stereo}
    SampleRate: Longword;                 {number of samples/sec}
    ByteRate: Longword;                   {samplerate* num of channels*bytes per (mono) sample}
    BytesperSample: Word;                 {size of (mono) sample}
    BitsPerSample: Word;                  {BytesperSample *8}
    SubChunk2ID: array[0..3] of AnsiChar; {'data'}
    SubChunk2Size: Longword;              {number of data bytes}
  end;
```

```
var
  vWavFormat1: TWavFormat;
  vWavFormat2: TWavFormat;
  vFileHandle1: THandle;
  vFileHandle2: THandle;
  vFileStream1: TFileStream;
  vFileStream2: TFileStream;
  vChunkSize1, vChunkSize2: Integer;
begin
  Result := False;
  if not FileExists(AWavFile1) then Exit;
  if not FileExists(AWavFile2) then Exit;

  vFileHandle1 := _lopen(PAnsiChar(AnsiString(AWavFile1)), OF_READ or OF_SHARE_DENY_NONE);
  vFileHandle2 := _lopen(PAnsiChar(AnsiString(AWavFile2)), OF_READ or OF_SHARE_DENY_NONE);

  if (Integer(vFileHandle1) <= 0) or (Integer(vFileHandle2) <= 0) then
  begin
    _lclose(vFileHandle1);
    _lclose(vFileHandle2);
    Exit;
  end;

  vFileStream1 := TFileStream.Create(vFileHandle1);
  vFileStream2 := TFileStream.Create(vFileHandle2);
  try
    if vFileStream1.Read(vWavFormat1, SizeOf(TWavFormat)) <>
SizeOf(TWavFormat) then Exit;
    if vFileStream2.Read(vWavFormat2, SizeOf(TWavFormat)) <>
SizeOf(TWavFormat) then Exit;
    if vWavFormat1.ChunkID <> 'RIFF' then Exit;
    if vWavFormat1.SubChunk2ID <> 'data' then Exit;
    vChunkSize1 := vWavFormat1.SubChunk2Size;
    vChunkSize2 := vWavFormat2.SubChunk2Size;
    vWavFormat1.ChunkSize := 0;
    vWavFormat1.SubChunk2Size := 0;
    vWavFormat2.ChunkSize := 0;
    vWavFormat2.SubChunk2Size := 0;
    if not CompareMem(@vWavFormat1, @vWavFormat2, SizeOf(TWavFormat))
then Exit; {格式不同}
    with TMemoryStream.Create do try
      vWavFormat1.ChunkSize := vChunkSize1 + vChunkSize2 +
SizeOf(vWavFormat1) - 8;
      vWavFormat1.SubChunk2Size := vChunkSize1 + vChunkSize2;
      Write(vWavFormat1, SizeOf(TWavFormat));
      CopyFrom(vFileStream1, vChunkSize1);
      CopyFrom(vFileStream2, vChunkSize2);
      try
        SaveToFile(ANewFile);
      except
        Exit;
      end;
    finally
      Free;
    end;
  finally
    vFileStream1.Free;
    vFileStream2.Free;
  end;
  Result := True;
end; { ConWavFile End}
```

```
{测试}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Wav1,Wav2,WavDest: string;
begin
  Wav1 := 'c:\temp\1.wav';
  Wav2 := 'c:\temp\2.wav';
  WavDest := 'c:\temp\12.wav';
  if ConWavFile(Wav1, Wav2, WavDest) then
    ShowMessageFmt('"%s" 和 "%s" 已成功合并到 "%s"', [Wav1,Wav2,WavDest]);
end;

end.
```

合并两个 Wav 文件流的函数 - 回复 "刘文强" 的问题

问题来源：http://www.cnblogs.com/del/archive/2008/10/25/1069523.html#1351197

```
 unit Unit1;

interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
uses MMSystem;


{合并两个 Wav 文件流的函数}
function ConWavStream(AWavStream1, AWavStream2: TStream; var ANewStream:
TStream): Boolean;
type
  TWavFormat = packed record
    ChunkID: array[0..3] of AnsiChar;      {'RIFF'}
    ChunkSize: Longword;                   {size-8}
    Format: array[0..3] of AnsiChar;       {'WAVE'}
    SubChunk1ID: array[0..3] of AnsiChar; {'fmt '}
    SubChunk1Size: Longword;               {hex10}
    AudioFormat: Word;                     {hex 01}
    NumOfChannels: Word;                   {1 mono, 2 stereo}
    SampleRate: Longword;                  {number of samples/sec}
    ByteRate: Longword;                    {samplerate* num of channels*bytes
per (mono) sample}
    BytesperSample: Word;                  {size of (mono) sample}
    BitsPerSample: Word;                   {BytesperSample *8}
    SubChunk2ID: array[0..3] of AnsiChar; {'data'}
    SubChunk2Size: Longword;               {number of data bytes}
  end;
```

```
var
  vWavFormat1: TWavFormat;
  vWavFormat2: TWavFormat;
  vChunkSize1, vChunkSize2: Integer;
begin
  Result := False;
    if AWavStream1.Read(vWavFormat1, SizeOf(TWavFormat)) <>
SizeOf(TWavFormat) then Exit;
    if AWavStream2.Read(vWavFormat2, SizeOf(TWavFormat)) <>
SizeOf(TWavFormat) then Exit;
  if vWavFormat1.ChunkID <> 'RIFF' then Exit;
  if vWavFormat1.SubChunk2ID <> 'data' then Exit;
  vChunkSize1 := vWavFormat1.SubChunk2Size;
  vChunkSize2 := vWavFormat2.SubChunk2Size;
  vWavFormat1.ChunkSize := 0;
  vWavFormat1.SubChunk2Size := 0;
  vWavFormat2.ChunkSize := 0;
  vWavFormat2.SubChunk2Size := 0;
  if not CompareMem(@vWavFormat1, @vWavFormat2, SizeOf(TWavFormat)) then
Exit; {格式不同}

   vWavFormat1.ChunkSize := vChunkSize1 + vChunkSize2 +
SizeOf(vWavFormat1) - 8;

  vWavFormat1.SubChunk2Size := vChunkSize1 + vChunkSize2;

  ANewStream.Write(vWavFormat1, SizeOf(TWavFormat));

  ANewStream.CopyFrom(AWavStream1, vChunkSize1);

  ANewStream.CopyFrom(AWavStream2, vChunkSize2);



  Result := True;

end; { ConWavStream End}





var

  WavStream: TStream;



{合并两个资源流; 之前要在资源中分别加载两个 WAV 文件, 并分别命名: wav1、wav2}

procedure TForm1.FormCreate(Sender: TObject);

var

  rs1,rs2: TResourceStream;

begin

  rs1 := TResourceStream.Create(HInstance, 'wav1', RT_RCDATA);

  rs2 := TResourceStream.Create(HInstance, 'wav2', RT_RCDATA);

  WavStream := TMemoryStream.Create;

  ConWavStream(rs1, rs2, WavStream);
  rs1.Free;
  rs2.Free;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  WavStream.Free;
end;

{循环播放}
procedure TForm1.Button1Click(Sender: TObject);
begin
  sndPlaySound(TMemoryStream(WavStream).Memory, SND_ASYNC or
SND_MEMORY or SND_LOOP);
end;

{暂停}
procedure TForm1.Button2Click(Sender: TObject);
begin
  sndPlaySound(nil, 0);
end;
```

end.

操作 Wave 文件(1): 关于 Wave 文件的基础知识与文件格式

最近准备学习 DirectSound、DirectMusic、DirectShow, 但刚一接触就碰到了关于 Wave 文件的诸多问题, 只好先回头学学 Wave 文件.

Wave 文件的基础知识

经常见到这样的描述: 44100HZ 16bit stereo 或者 22050HZ 8bit mono 等等.

44100HZ 16bit stereo : 每秒钟有 44100 次采样, 采样数据用 16 位(2 字节)记录, 双声道(立体声);

22050HZ 8bit mono : 每秒钟有 22050 次采样, 采样数据用 8 位(1 字节)记录, 单声道;

当然也可以有 16bit 的单声道或 8bit 的立体声, 等等.

人对频率的识别范围是 20HZ - 20000HZ, 如果每秒钟能对声音做 20000 个采样, 回放时就足可以满足人耳的需求. 所以 22050 的采样频率是常用的, 44100 已是 CD 音质, 超过 48000 的采样对人耳已经没有意义. 这和电影的每秒 24 帧图片的道理差不多.

每个采样数据记录的是振幅, 采样精度取决于储存空间的大小:

1 字节(也就是 8bit) 只能记录 256 个数, 也就是只能对振幅做 256 种识别;

2 字节(也就是 16bit) 可以细到 65536 个数, 这已是 CD 标准了;

4 字节(也就是 32bit) 能把振幅细化到 4294967296 种可能性, 实在是没必要了.

如果是双声道(stereo), 采样就是双份的, 文件也差不多要大一倍.

这样我们就可以根据一个 wav 文件的大小、采样频率和采样大小估算出一个 wav 文件的长度; 譬如 "Windows XP 启动.wav" 的文件长度是 424,644 字节, 它是 "22050HZ / 16bit / 立体声" 格式(这可以从其 "属性->摘要" 里看到).

它的每秒的传输速率是 22050*16*2 = 705600(bit), 换算成字节是 705600/8 = 88200(字节);

424644(总字节数) / 88200(每秒字节数) ≈ 4.8145578(秒).

这还不够精确, 在标准的 PCM 格式的 WAVE 文件中还有 44 个字节是采样数据之外的

内容，应该去掉:

(424644-44) / (22050*16*2/8) ≈ 4.8140589(秒). 这比较精确了.

关于声音文件还有一个概念: "位速", 也有叫做比特率、取样率, 譬如上面文件的位速是 705.6kbps 或 705600bps, 其中的 b 是 bit, ps 是每秒的意思; 压缩的音频文件常常用位速来表示, 譬如达到 CD 音质的 mp3 是: 128kbps / 44100HZ.

Wave 文件的文件格式

微软的多媒体文件(wav、avi、tif 等)都有一个 RIFF 头, Wave 文件基本是这个样子:

RIFF 头	
	fmt 子块
	data 子块

Wave 文件的编码方式有好多, 最常用最简单的就是 PCM 编码.

其他编码会包含更多的"块", 但至少会包含上面的块, PCM 编码只包含上面的块.

下面是 PCM 编码的样表:

RIFF 头	ckid	4	"RIFF" 标识			
	cksize	4	文件大小; 这个大小不包括 ckid 和 cksize 本身, 下面的子块大小也是这样			
	fccType	4	类型, 这里是 "WAVE" 标识			
		24	fmt 子块	ckid	4	"fmt " 标识
				cksize	4	块大小; 对 PCM 编码这里是 16, 其他编码也不小于 16
				wFormatTag	2	编码格式; 1 表示是 PCM 编码
				nChannels	2	声道数; 1 是单声道、2 是立体声
				nSamplesPerSec	4	采样频率(每秒的样本数); 譬如 44100
				nAvgBytesPerSec	4	传输速率 = 采样频率 * 每次采样大小, 单位是字节
				nBlockAlign	2	每次采样的大小 = 采样精度 * 声道数 / 8(因单位是字节所以要/8); 这也是字节对齐的最小单位, 譬如 16bit 立体声在这里的值是 4 字节
				wBitsPerSample	2	采样精度; 譬如 16bit 在这里的值就是 16
		?	data 子块	ckid	4	"data" 标识
				cksize	4	块大小
				采样数据	?	双声道数据排列: 左右左右...; 8bit: 0-255, 16bit: -32768-32767

其他编码可能会包含的块有: 事实块(Fact)、提示块(Cue)、标签块(Label)、注释块(Note)、标签文本块(Labeled Text)、采样器块(Sampler)、乐器块(Instrument)、列表块(List)等等, 如果有 List 块, 它还会包含更多子块.

接下来要存取、播放、录制，说来容易，操作起来都挺麻烦.

操作 Wave 文件(2)：判断一个文件是否是 Wave 文件

Wave 文件的前 12 个字节可以这样描述:

```
---------------------------------------------------------------------------------

 TRiff = record
   ckId      : DWORD; {'RIFF'}
   ckSize    : DWORD; {文件大小，不包括前 8 个字节}
   fccType : DWORD; {'WAVE'}
end;
---------------------------------------------------------------------------------
```

我们读出文件的前 12 个字节进行判断，就基本可以确认它是不是 Wave 文件.

```
---------------------------------------------------------------------------------

 uses MMSystem, IOUtils; {这里准备用 IOUtils.TFile.OpenRead 方便地建立文件流}
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
  riff: record ckId, ckSize, fccType: DWORD; end; {可以同时定义结构并声明结构变量}
begin
  with TFile.OpenRead('C:\WINDOWS\Media\Windows XP 启动.wav') do
  begin
    Read(riff, SizeOf(riff));
    Free;
  end;

    if (riff.ckId = FOURCC_RIFF) and (riff.fccType = mmioStringToFOURCC('WAVE',0)) then
    ShowMessageFmt('这是个 Wave 文件, 其大小是 %d 字节', [riff.ckSize + 8]);
end;
```

还是把它写成一个函数吧, 最好也别再引用 MMSystem 单元.

```
 {如果是 Wave 文件则返回文件大小，不是则返回 0}
function IsWave(FilePath: string): Integer;
  function mmioFOURCC(Chr0,Chr1,Chr2,Chr3: AnsiChar): DWORD;
  begin
      Result := DWORD(Chr0) + DWORD(Chr1) shl 8 + DWORD(Chr2) shl 16 +
DWORD(Chr3) shl 24;
  end;
var
  riff: record ckId, ckSize, fccType: DWORD; end;
begin
  Result := 0;
  with TFileStream.Create(FilePath, fmOpenRead) do begin
    Read(riff, SizeOf(riff));
    Free;
  end;
  if (riff.ckId = mmioFOURCC('R', 'I', 'F', 'F')) and
     (riff.fccType = mmioFOURCC('W', 'A', 'V', 'E')) then
    Result := riff.ckSize + 8;
end;
```


依次道理，也可以判断一个 RIFF 文件具体是什么格式.

```
 {返回 RIFF 文件格式的函数，如果不是 RIFF 文件，则返回 'noneRIFF'}
function GetRiffType(FilePath: string): String;
  function mmioFOURCC(Chr0,Chr1,Chr2,Chr3: AnsiChar): DWORD;
  begin
      Result := DWORD(Chr0) + DWORD(Chr1) shl 8 + DWORD(Chr2) shl 16 + DWORD(Chr3) shl 24;
  end;
var
  riff: record ckId, ckSize, fccType: DWORD; end;
type
  TChars = array[0..3] of AnsiChar; {用于类型转换}
begin
  Result := 'noneRIFF';
  with TFileStream.Create(FilePath, fmOpenRead) do begin
    Read(riff, SizeOf(riff));
    Free;
  end;
  if (riff.ckId = mmioFOURCC('R', 'I', 'F', 'F')) then Result := TChars(riff.fccType);
end;


//测试:
begin
    ShowMessage(GetRiffType('C:\WINDOWS\Media\Windows XP 启动.wav'));
{WAVE}
  ShowMessage(GetRiffType('C:\WINDOWS\clock.avi'));                {AVI }
                         ShowMessage(GetRiffType('C:\WINDOWS\notepad.exe'));
{noneRIFF}
end;
------------------------------------------------------------------------------------
```

关于 FOURCC_RIFF、mmioFOURCC、mmioStringToFOURCC:

RIFF 格式的文件都是有若干 "块" 来构成的, 每个块都是有 4 个字符开头(不足 4 个字符用空格补足);

这连续的 4 个字节刚好是一个 32 位整数的大小, 所以常常把它们当作一个整数读出来判断.

通过 MMSystem.mmioStringToFOURCC 就可以获取这样的整数.

从 C/C++ 代码中经常看到: mmioFOURCC; 它并非 winmm.dll 库中的函数, 是在 C/C++ 中定义的宏.

这里用 Delphi 模拟实现了这个函数. 其功能类似 mmioStringToFOURCC.

MMSystem.FOURCC_RIFF 是个常量, 当需要 "RIFF" 对应的整数时直接用就是了. 举例:

```
 uses MMSystem;

{自定义的 mmioFOURCC 函数}
function mmioFOURCC(Chr0,Chr1,Chr2,Chr3: AnsiChar): DWORD;
begin
   Result := DWORD(Chr0) + DWORD(Chr1) shl 8 + DWORD(Chr2) shl 16 +
DWORD(Chr3) shl 24;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  f1,f2,f3,f4: FOURCC; {FOURCC = DWORD;}
begin
```

```
  f1 := mmioStringToFOURCC('RIFF', 0);
  f2 := mmioStringToFOURCC('Riff', MMIO_TOUPPER); {第二个参数可以把字符串转大写}

  f3 := mmioFOURCC('R', 'I', 'F', 'F');

  f4 := FOURCC_RIFF;

  ShowMessageFmt('%d, %d, %d, %d', [f1,f2,f3,f4]);
  {1179011410, 1179011410, 1179011410, 1179011410}
end;
```

操作 Wave 文件(3): 接触 mmio 系列函数

mmio 系列函数用于 Wave 等多媒体文件的 I/O 操作，相关函数有:

----

mmioOpen

mmioClose

mmioRead

mmioWrite

mmioFlush

mmioSeek

mmioRename

mmioGetInfo

mmioSetInfo

mmioCreateChunk

mmioAscend

mmioDescend

mmioAdvance

mmioSetBuffer

mmioStringToFOURCC

mmioSendMessage

mmioInstallIOProc

----

mmio 系列函数比一般的 I/O 函数更适合操作 RIFF 格式的多媒体文件，主要是能更方便地操作 RIFF 的文件块，官方还说它们更优化.

和其他 I/O 函数一样，它们也是要 Open 获取句柄，然后读写，最后关闭；但它们文件句柄和其他 I/O 函数的句柄并不兼容，不过部分函数(上面前 7 个)也可以用于一般文件的操作.

就先操作个一般文件吧.

```
uses MMSystem;

procedure TForm1.FormCreate(Sender: TObject);
const
  FilePath = 'C:\Temp\mm.txt';
var
  hFile: HMMIO;
  str: RawByteString;
begin
  hFile := mmioOpen(PChar(FilePath),              {要打开的文件}
                    nil,                          {接受 TMMIOInfo 结构信息的指针，暂时没用到}
                    MMIO_CREATE or MMIO_READWRITE {打开选项；这是建立并以
读写权限打开}
                         );
  mmioWrite(hFile, 'Delphi', 6);        {写入 6 个字符}

  mmioSeek(hFile, 0, SEEK_SET);         {把读写指针移动到文件头}

  SetLength(str, 6);
  mmioRead(hFile, PAnsiChar(str), 6); {读出 6 个字符}
  ShowMessage(str);                        {Delphi}

   mmioClose(hFile，0);                              {关闭文件；第二个参数还可以是
MMIO_FHOPEN，另有它用}

  {最后再删除这个文件，既然已删除就无需 Close 了}
  mmioOpen(PChar(FilePath), nil, MMIO_DELETE);
end;
```

操作 Wave 文件(4)：获取 Wave 文件主块与子块的信息

有两个相关的结构体: TMMIOInfo、TMMCKInfo.

TMMIOInfo是多媒体文件打开后的状态信息, mmioOpen函数的第二个参数就是这个结构的指针.

现在先用到了TMMCKInfo, 这是文件内部"块"的信息, 构成如下:

```
TMMCKInfo = record
  ckid: FOURCC;          {块标识}
  cksize: DWORD;          {块大小}
  fccType: FOURCC;       {格式类型标识}
  dwDataOffset: DWORD; {偏移地址}
  dwFlags: DWORD;        {附加信息}
end;
```

查找"块"需要通过mmioDescend、mmioAscend两个函数.

mmioAscend是从子块跳出;

mmioDescend是进入到子块; 进入子块是需要指定子块的ckid和父块信息;

mmioDescend也用来查找主块(RIFF), 此时需要很少的信息就可以找到主块.

测试代码:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
uses MMSystem;

const FilePath = 'C:\WINDOWS\Media\Windows XP 启动.wav';

//获取 RIFF 块的信息
procedure TForm1.Button1Click(Sender: TObject);
var
  hFile: HMMIO;
  ckiRIFF: TMMCKInfo;
begin
  //清空 ckiRIFF 结构体; 有些函数使用前要求必须清空, 即使不要求也还是清空的好.
  FillChar(ckiRIFF, SizeOf(TMMCKInfo), 0); {局部变量在清空前有垃圾数据}

  //打开文件, 获取句柄
  hFile := mmioOpen(PChar(FilePath), nil, MMIO_READ);

  //获取 RIFF 块的信息
  mmioDescend(hFile,          {文件句柄}
              @ckiRIFF,       {块信息结构的指针, 用于获取块的信息}
              nil,            {这父块的结构信息, RIFF 没有父块, 无需指定}
              MMIO_FINDRIFF {如果是查询子块这里的标志是 MMIO_FINDCHUNK}
```

```
            );              {返回 0 表示查找成功, 这里忽略了验证}

  //以下是查证获取到的信息
    ShowMessageFmt('%d, %d, %d, %d, %d', [ckiRIFF.ckid, ckiRIFF.cksize, ckiRIFF.fccType,
  ckiRIFF.dwDataOffset, ckiRIFF.dwFlags ]); {1179011410, 424636, 1163280727, 8, 0}

  if ckiRIFF.ckid = FOURCC_RIFF then ShowMessage('是 RIFF');
   if ckiRIFF.fccType = mmioStringToFOURCC('WAVE',0) then ShowMessage('是 WAVE');

  //关闭
  mmioClose(hFile, 0);
end;


//获取子块的信息
procedure TForm1.Button2Click(Sender: TObject);
var
  hFile: HMMIO;
  ckiRIFF,ckiSub: TMMCKInfo;
  n: Integer;
begin
  //清空准备接受信息的结构
  FillChar(ckiRIFF, SizeOf(TMMCKInfo), 0);
  FillChar(ckiSub, SizeOf(TMMCKInfo), 0);

  hFile := mmioOpen(PChar(FilePath), nil, MMIO_READ);

  //先获取主块(RIFF)信息
  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);

  //获取 fmt 子块信息
  ckiSub.ckid := mmioStringToFOURCC('fmt', 0);
      if mmioDescend(hFile, @ckiSub, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR then
  begin
          ShowMessageFmt('%d, %d, %d, %d, %d', [ckiSub.ckid, ckiSub.cksize,
ckiSub.fccType,
      ckiSub.dwDataOffset, ckiSub.dwFlags]);
  end;

  //如果继续查找需要跳出子块; 下面将从偏移地址 20 跳到 36 处
  mmioAscend(hFile, @ckiSub, 0); {其第三个参数一直是 0，是备用参数}

  //获取 data 子块信息
  ckiSub.ckid := mmioStringToFOURCC('data', 0);
    if mmioDescend(hFile, @ckiSub, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR then
  begin
      ShowMessageFmt('%d, %d, %d, %d, %d', [ckiSub.ckid, ckiSub.cksize,
ckiSub.fccType,
    ckiSub.dwDataOffset, ckiSub.dwFlags]);
  end;

  mmioClose(hFile, 0);
end;

end.
```

操作 Wave 文件(5)：获取 Wave 文件的格式信息

装载格式信息的结构有:

```
TWaveFormat = packed record
  wFormatTag: Word;
  nChannels: Word;
  nSamplesPerSec: DWORD;
  nAvgBytesPerSec: DWORD;
  nBlockAlign: Word;
end;

TPCMWaveFormat = record
  wf: TWaveFormat;
  wBitsPerSample: Word;
end;

TWaveFormatEx = packed record
  wFormatTag: Word;       {格式类型; 主要使用的是 WAVE_FORMAT_PCM}
  nChannels: Word;        {声道数; 1 是单声道、2 是立体声}
  nSamplesPerSec: DWORD;  {采样频率}
  nAvgBytesPerSec: DWORD; {传输速率}
  nBlockAlign: Word;      {每次采样的大小}
  wBitsPerSample: Word;   {采样精度}
  cbSize: Word;           {附加数据的大小; PCM 编码的文件没这个字段}
end;
```

能看出它们是依次递增一个字段，并且也是 Wave 文件的一个构成部分；现在要做的就是从 Wave 文件中把它们取出来.

获取函数及测试代码:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

//获取Wave中格式数据的函数; 常用的是TWaveFormatEx, 但PCM缺它一个字段
function GetWaveFmt(FilePath: string; var fmt: TWaveFormatEx): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiFmt: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(FilePath), nil, MMIO_READ);
  if hFile = 0 then Exit;

  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));      {也先清空准备接受数据的结构体}
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0); {给查找格式块准备}

  //先获取主块的信息
  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);

  //再获取 fmt 块的信息后，指针将自动指向格式数据起点；然后读出格式数据
  if (ckiRIFF.ckid = FOURCC_RIFF) and
     (ckiRIFF.fccType = mmioStringToFOURCC('WAVE',0)) and
        (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) = MMSYSERR_NOERROR) then
     Result := (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize);
  mmioClose(hFile, 0);
end;

//调用测试
procedure TForm1.Button1Click(Sender: TObject);
const
  FilePath = 'C:\WINDOWS\Media\Windows XP 启动.wav';
var
  WaveFormat: TWaveFormatEx;
```

```
begin
  if GetWaveFmt(FilePath, WaveFormat) then with Memo1.Lines do
  begin
    Clear;
    Add(Format('wFormatTag: %d', [WaveFormat.wFormatTag]));
    Add(Format('nChannels: %d', [WaveFormat.nChannels]));
    Add(Format('nSamplesPerSec: %d', [WaveFormat.nSamplesPerSec]));
    Add(Format('nAvgBytesPerSec: %d', [WaveFormat.nAvgBytesPerSec]));
    Add(Format('nBlockAlign: %d', [WaveFormat.nBlockAlign]));
    Add(Format('wBitsPerSample: %d', [WaveFormat.wBitsPerSample]));
    Add(Format('cbSize: %d', [WaveFormat.cbSize]));
  end;
{ 显示结果:
  wFormatTag: 1
  nChannels: 2
  nSamplesPerSec: 22050
  nAvgBytesPerSec: 88200
  nBlockAlign: 4
  wBitsPerSample: 16
  cbSize: 0
}
end;
```

```
end.
```

操作 Wave 文件(6): 获取 Wave 文件的波形数据

读取函数及测试代码:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}


uses MMSystem;

function GetWaveData(FilePath: string; var stream: TMemoryStream): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(FilePath), nil, MMIO_READ);
  if hFile = 0 then Exit;

  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));
  ckiData.ckid := mmioStringToFOURCC('data', 0);

  //先获取主块的信息
  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);

  //再获取 data 块的信息后，指针将自动指向 data 数据的起点；然后读出数据
  if (ckiRIFF.ckid = FOURCC_RIFF) and
     (ckiRIFF.fccType = mmioStringToFOURCC('WAVE',0)) and
            (mmioDescend(hFile,   @ckiData,   @ckiRIFF,   MMIO_FINDCHUNK)   =
MMSYSERR_NOERROR) then
    begin
      stream.Size := ckiData.cksize;
              Result   :=   (mmioRead(hFile,   stream.Memory,   ckiData.cksize)   =
ckiData.cksize);
    end;
  mmioClose(hFile, 0);
end;

//调用测试
procedure TForm1.Button1Click(Sender: TObject);
const
  FilePath = 'C:\WINDOWS\Media\Windows XP 启动.wav';
```

```
var
  stream: TMemoryStream;
begin
  stream := TMemoryStream.Create;

  if GetWaveData(FilePath, stream) then
    ShowMessageFmt('读出的数据大小是: %d', [stream.Size]); {424600}

  stream.Free;
end;

end.
```

操作 Wave 文件(7): 建立一个空的 Wave 文件(三种方法)

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

//chan: 1 单声道、2 立体声;
//freq: 频率，取值: 11025, 22050, 44100
//bit : 每个样本的大小，取值 8、16
function CreateWav1(chan, freq, bit: Word; const FilePath: string): Boolean;
var
  h: HMMIO;
  ckiRiff, ckiFmt, ckiData: TMMCKInfo;
  fmt: TPCMWaveFormat;
begin
  //此函数是使用 mmioCreateChunk 函数来分别建立 Wave 文件的每个块.

  {初识化相关结构}
  ZeroMemory(@ckiRiff, SizeOf(TMMCKInfo));
  ckiRiff.cksize := 36; {mmioCreateChunk 函数会自动写上 ckid, 但其 cksize 需要手动给}
  ckiRiff.fccType := mmioStringToFOURCC('WAVE', 0);

  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);

  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));
  ckiData.ckid := mmioStringToFOURCC('data', 0);

  {指定 Wave 格式}
  fmt.wf.wFormatTag := WAVE_FORMAT_PCM;
  fmt.wf.nChannels := chan;
  fmt.wf.nSamplesPerSec := freq;
  fmt.wf.nAvgBytesPerSec := freq * chan * bit div 8;
  fmt.wf.nBlockAlign := chan * bit div 8;
  fmt.wBitsPerSample := bit;

  h := mmioOpen(PChar(FilePath), nil, MMIO_CREATE or MMIO_WRITE);
  if h = 0 then Exit(False);

  {分别建立 RIFF、fmt、data 块}
  if (mmioCreateChunk(h, @ckiRiff, MMIO_CREATERIFF) = MMSYSERR_NOERROR)
and
    (mmioCreateChunk(h, @ckiFmt, 0) = MMSYSERR_NOERROR) and
            (mmioWrite(h,   PAnsiChar(@fmt),   SizeOf(TPCMWaveFormat))   =
SizeOf(TPCMWaveFormat)) and
    (mmioAscend(h, @ckiFmt, 0) = MMSYSERR_NOERROR) and
     (mmioCreateChunk(h, @ckiData, 0) = MMSYSERR_NOERROR) then Result :=
True;

  mmioClose(h, 0);
end;

//把 PCM 编码的 WAVE 文件的前 44 个字节看成一个结构来操作:
function CreateWav2(chan, freq, bit: Word; const FilePath: string): Boolean;
type
```

```
  TWaveHeader = record
    Riff_ckid        : DWORD;
    Riff_cksize      : DWORD;
    Riff_fccType     : DWORD;
    fmt_ckid         : DWORD;
    fmt_cksize       : DWORD;
    wFormatTag       : Word;
    nChannels        : Word;
    nSamplesPerSec : DWORD;
    nAvgBytesPerSec: DWORD;
    nBlockAlign      : Word;
    wBitsPerSample : Word;
    data_ckid        : DWORD;
    data_cksize      : DWORD;
  end;
var
  wh: TWaveHeader;
  hFile: Integer;
begin
  wh.Riff_ckid := FOURCC_RIFF;
  wh.Riff_cksize := 36;
  wh.Riff_fccType := mmioStringToFOURCC('WAVE', 0);
  wh.fmt_ckid := mmioStringToFOURCC('fmt', 0);

  wh.fmt_cksize := 16;

  wh.wFormatTag := WAVE_FORMAT_PCM;

  wh.nChannels := chan;

  wh.nSamplesPerSec := freq;

  wh.nAvgBytesPerSec := freq * chan * bit div 8;

  wh.nBlockAlign := chan * bit div 8;

  wh.wBitsPerSample := bit;

  wh.data_ckid := mmioStringToFOURCC('data', 0);

  wh.data_cksize := 0;


  hFile := FileCreate(FilePath);

  Result := (FileWrite(hFile, wh, SizeOf(TWaveHeader)) <> -1);

  FileClose(hFile);

end;


//同上，只是改用流来写文件

function CreateWav3(chan, freq, bit: Word; const FilePath: string): Boolean;

type

  TWaveHeader = record

    Riff_ckid       : DWORD;

    Riff_cksize     : DWORD;
    Riff_fccType    : DWORD;
    fmt_ckid        : DWORD;
    fmt_cksize      : DWORD;
    wFormatTag      : Word;
    nChannels       : Word;
    nSamplesPerSec  : DWORD;
    nAvgBytesPerSec: DWORD;
    nBlockAlign     : Word;
    wBitsPerSample : Word;
    data_ckid       : DWORD;
    data_cksize     : DWORD;
  end;
var
  wh: TWaveHeader;
begin
  wh.Riff_ckid := FOURCC_RIFF;
  wh.Riff_cksize := 36;
  wh.Riff_fccType := mmioStringToFOURCC('WAVE', 0);
  wh.fmt_ckid := mmioStringToFOURCC('fmt', 0);
  wh.fmt_cksize := 16;
  wh.wFormatTag := WAVE_FORMAT_PCM;
  wh.nChannels := chan;
  wh.nSamplesPerSec := freq;

  wh.nAvgBytesPerSec := freq * chan * bit div 8;

  wh.nBlockAlign := chan * bit div 8;

  wh.wBitsPerSample := bit;

  wh.data_ckid := mmioStringToFOURCC('data', 0);

  wh.data_cksize := 0;


  with TFileStream.Create(FilePath, fmCreate) do begin

    Result := (Write(wh, SizeOf(TWaveHeader)) = SizeOf(TWaveHeader));

    Free;

  end;

end;


procedure TForm1.FormCreate(Sender: TObject);

begin

  CreateWav1(1, 11025, 8,   'C:\Temp\X1.wav');

  CreateWav2(2, 22050, 16, 'C:\Temp\X2.wav');

  CreateWav3(2, 44100, 16, 'C:\Temp\X3.wav');

end;


end.
```

操作 Wave 文件(8): 使用 TMediaPlayer 录制 wav 文件

TMediaPlayer 录音是基于一个已存在的 wav 文件, 上次建立空白 wav 的函数可派上用场了.

TMediaPlayer 的功能是基于 MCI 的, 都是该淘汰的东西了, 只是简单了解下.

接下来还要学习用 waveIn...系列函数录音、用 DirectSound 录音.

```
unit Unit1;



interface


uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, MPlayer, StdCtrls;


type
  TForm1 = class(TForm)
    MediaPlayer1: TMediaPlayer;
    Button1: TButton;

    Button2: TButton;

    procedure FormCreate(Sender: TObject);

    procedure Button1Click(Sender: TObject);

    procedure Button2Click(Sender: TObject);

  end;


var

  Form1: TForm1;


implementation


{$R *.dfm}


uses MMSystem;


//建立一个空白 Wave 文件的函数

function CreateWav(chan, freq, bit: Word; const FilePath: string): Boolean;

var

  h: HMMIO;

  ckiRiff, ckiFmt, ckiData: TMMCKInfo;

  fmt: TPCMWaveFormat;
```

```
begin
  ZeroMemory(@ckiRiff, SizeOf(TMMCKInfo));
  ckiRiff.cksize := 36;
  ckiRiff.fccType := mmioStringToFOURCC('WAVE', 0);

  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);

  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));
  ckiData.ckid := mmioStringToFOURCC('data', 0);

  fmt.wf.wFormatTag := WAVE_FORMAT_PCM;
  fmt.wf.nChannels := chan;
  fmt.wf.nSamplesPerSec := freq;
  fmt.wf.nAvgBytesPerSec := freq * chan * bit div 8;
  fmt.wf.nBlockAlign := chan * bit div 8;
  fmt.wBitsPerSample := bit;

  h := mmioOpen(PChar(FilePath), nil, MMIO_CREATE or MMIO_WRITE);
  if h = 0 then Exit(False);

  if (mmioCreateChunk(h, @ckiRiff, MMIO_CREATERIFF) = MMSYSERR_NOERROR)
```

```
and
    (mmioCreateChunk(h, @ckiFmt, 0) = MMSYSERR_NOERROR) and
             (mmioWrite(h,   PAnsiChar(@fmt),   SizeOf(TPCMWaveFormat))   =
SizeOf(TPCMWaveFormat)) and
    (mmioAscend(h, @ckiFmt, 0) = MMSYSERR_NOERROR) and
     (mmioCreateChunk(h, @ckiData, 0) = MMSYSERR_NOERROR) then Result :=
True;

  mmioClose(h, 0);
end;

//文件路径
const path = 'C:\Temp\Test.wav';

//开始录音
procedure TForm1.Button1Click(Sender: TObject);
begin
  CreateWav(2, 22050, 16, path);

  MediaPlayer1.FileName := path;
  MediaPlayer1.Open;
  MediaPlayer1.StartRecording;
  Button2.Enabled := True;
end;

//停止录音并播放
procedure TForm1.Button2Click(Sender: TObject);
begin
  MediaPlayer1.Stop;
  MediaPlayer1.Play;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  MediaPlayer1.Visible := False;
  Button2.Enabled := FileExists(path);
end;

end.
```

操作 Wave 文件(9): 使用 waveOut... 函数播放 wav 文件

下面是使用低级音频函数播放 wav 的两个方法, 对这个感兴趣的人恐怕很少, 免注释了.

使用窗口接受音频输出设备的消息:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;


type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
  protected
    procedure WndProc(var Message: TMessage); override;
  public
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

//获取文件格式和波形数据的函数
function GetWaveFmtData(path: string; var fmt: TWaveFormatEx; var buf: TBytes): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiFmt,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(path), nil, MMIO_READ);
  if hFile = 0 then Exit;

  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));

  ckiRIFF.fccType := mmioStringToFOURCC('WAVE', 0);
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);
  ckiData.ckid := mmioStringToFOURCC('data', 0);

  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));

  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);

      if (ckiRIFF.ckid = FOURCC_RIFF) and (ckiRIFF.fccType = mmioStringToFOURCC('WAVE',0)) and
          (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) = MMSYSERR_NOERROR) and
     (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize) and
     (mmioAscend(hFile, @ckiFmt, 0) = MMSYSERR_NOERROR) and
          (mmioDescend(hFile, @ckiData, @ckiRIFF, MMIO_FINDCHUNK) = MMSYSERR_NOERROR) then
  begin
    SetLength(buf, ckiData.cksize);
    Result := (mmioRead(hFile, PAnsiChar(buf), ckiData.cksize) = ckiData.cksize);
  end;


  mmioClose(hFile, 0);
end;


//---------------------------------------------------------------------------
------------
var
  wh: TWaveHdr;
  hOut: HWAVEOUT;
  fmt: TWaveFormatEx;
  buf: TBytes;

procedure TForm1.Button1Click(Sender: TObject);
const
  path = 'C:\WINDOWS\Media\Windows XP 启动.wav';
begin
  GetWaveFmtData(path, fmt, buf);


  wh.lpData := PAnsiChar(buf);
  wh.dwBufferLength := Length(buf);
  wh.dwBytesRecorded := 0;
  wh.dwUser := 0;
  wh.dwFlags := 0;
  wh.dwLoops := 1;
  wh.lpNext := nil;
  wh.reserved := 0;

  waveOutOpen(@hOut, WAVE_MAPPER, @fmt, Handle, 0, CALLBACK_WINDOW);
  waveOutPrepareHeader(hOut, @wh, SizeOf(TWaveHdr));
  waveOutWrite(hOut, @wh, SizeOf(TWaveHdr));
end;

procedure TForm1.WndProc(var Message: TMessage);
begin
  inherited;
  case Message.Msg of
    MM_WOM_OPEN: ;
    MM_WOM_CLOSE: ;
    MM_WOM_DONE: begin
      waveOutUnprepareHeader(hOut, @wh, SizeOf(TWaveHdr));
      waveOutClose(hOut);
    end;
  end;
```

```
end;

end.
```

使用回调函数:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

function GetWaveFmtData(path: string; var fmt: TWaveFormatEx; var buf: TBytes): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiFmt,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(path), nil, MMIO_READ);
  if hFile = 0 then Exit;
  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));

  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));

  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));


  ckiRIFF.fccType := mmioStringToFOURCC('WAVE', 0);

  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);

  ckiData.ckid := mmioStringToFOURCC('data', 0);


  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));


  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);


      if (ckiRIFF.ckid = FOURCC_RIFF) and (ckiRIFF.fccType =
mmioStringToFOURCC('WAVE',0)) and

        (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) and

     (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize) and

     (mmioAscend(hFile, @ckiFmt, 0) = MMSYSERR_NOERROR) and

        (mmioDescend(hFile, @ckiData, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) then

  begin
```

```
    SetLength(buf, ckiData.cksize);

    Result := (mmioRead(hFile, PAnsiChar(buf), ckiData.cksize) = ckiData.cksize);

  end;


  mmioClose(hFile, 0);

end;


//------------------------------------------------------------------

------------

var

  wh: TWaveHdr;

  hOut: HWAVEOUT;

  fmt: TWaveFormatEx;

  buf: TBytes;


procedure WaveProc(hWave: HWAVE; uMsg, dwInstance, dwParam1, dwParam2:

DWORD); stdcall;

begin

  case uMsg of

    MM_WOM_OPEN: ;

    MM_WOM_CLOSE: ;

    MM_WOM_DONE: begin
```

```
                    waveOutUnprepareHeader(hWave,    PWaveHdr(dwParam1),
SizeOf(TWaveHdr));
      waveOutClose(hWave);
    end;
  end;
end;

procedure TForm1.Button1Click(Sender: TObject);
const
  path = 'C:\WINDOWS\Media\Windows XP 启动.wav';
begin
  GetWaveFmtData(path, fmt, buf);

  wh.lpData := PAnsiChar(buf);
  wh.dwBufferLength := Length(buf);
  wh.dwBytesRecorded := 0;
  wh.dwUser := 0;
  wh.dwFlags := 0;
  wh.dwLoops := 1;
  wh.lpNext := nil;
  wh.reserved := 0;
```

```
    waveOutOpen(@hOut, WAVE_MAPPER, @fmt, DWORD(@WaveProc), 0,
CALLBACK_FUNCTION);
  waveOutPrepareHeader(hOut, @wh, SizeOf(TWaveHdr));
  waveOutWrite(hOut, @wh, SizeOf(TWaveHdr));
end;

//暂停
procedure TForm1.Button2Click(Sender: TObject);
begin
  waveOutPause(hOut);
end;

//继续
procedure TForm1.Button3Click(Sender: TObject);
begin
  waveOutRestart(hOut);
end;


end.
```

操作 Wave 文件(10): 输入输出设备与格式支持

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    ListBox2: TListBox;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  end;

var
```

```
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

//设备列表; 指定设备时经常使用 WAVE_MAPPER 参数, 这样会自动选用合适的设备.
procedure TForm1.Button1Click(Sender: TObject);
var
  i: Integer;
  waveOutCaps: TWaveOutCaps;
  waveInCaps: TWaveInCaps;
begin
  ListBox1.Items.Add('音频输出设备列表:');
  for i := 0 to waveOutGetNumDevs do
  begin
    ZeroMemory(@waveOutCaps, SizeOf(TWaveOutCaps));
    waveOutGetDevCaps(i, @waveOutCaps, SizeOf(TWaveOutCaps));
    ListBox1.Items.Add(waveOutCaps.szPname);
  end;
  ListBox2.Items.Add('音频输入设备列表:');
  for i := 0 to waveInGetNumDevs do
  begin
    ZeroMemory(@waveInCaps, SizeOf(TWaveInCaps));
    waveOutGetDevCaps(i, @waveInCaps, SizeOf(TWaveInCaps));
    ListBox2.Items.Add(waveInCaps.szPname);
  end;
end;

//判断是否支持指定的 Wave 格式
procedure TForm1.Button2Click(Sender: TObject);
var
  fmt: TPCMWaveFormat;
begin
  fmt.wf.wFormatTag := WAVE_FORMAT_PCM;
  fmt.wf.nChannels := 2;
  fmt.wf.nSamplesPerSec := 22050;
  fmt.wf.nAvgBytesPerSec := 88200;
  fmt.wf.nBlockAlign := 4;
  fmt.wBitsPerSample := 16;
  if waveOutOpen(nil, 0, PWaveFormatEx(@fmt), 0, 0, WAVE_FORMAT_QUERY) = 0 then
    ShowMessage('第一个输出设备支持此格式');

  if waveInOpen(nil, 0, PWaveFormatEx(@fmt), 0, 0, WAVE_FORMAT_QUERY) = 0 then
    ShowMessage('第一个输入设备支持此格式');
end;

end.
```

有把格式支持的判断写成函数的，如:

```
function IsFormatSupported(fmt: Pointer; DeviceId: DWORD): Boolean;
begin
  Result := (waveOutOpen(nil, DeviceId, PWaveFormatEx(fmt), 0, 0, WAVE_FORMAT_QUERY) = 0);
end;
```

操作 Wave 文件(11): 使用 waveIn...函数录制 wav 文件

使用窗口接受音频设备发出的消息:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  protected
    procedure WndProc(var m: TMessage); override;
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

var
  whIn1,whIn2, whOut: TWaveHdr;
  hWaveIn,hWaveOut: HWAVE;
  fmt: TWaveFormatEx;
  buf1,buf2,SaveBuf: TBytes;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := '开始录音';
  Button2.Caption := '停止录音';
  Button3.Caption := '播放录音';
end;

//开始录音
procedure TForm1.Button1Click(Sender: TObject);
begin
  {指定要录制的格式}
  fmt.wFormatTag := WAVE_FORMAT_PCM;
  fmt.nChannels := 2;
  fmt.nSamplesPerSec := 22050;
  fmt.nAvgBytesPerSec := 88200;
  fmt.nBlockAlign := 4;
  fmt.wBitsPerSample := 16;
  fmt.cbSize := 0;

  SaveBuf := nil; {清除已录制的内容}

  if waveInOpen(@hWaveIn, WAVE_MAPPER, @fmt, Handle, 0, CALLBACK_WINDOW) = 0 then
  begin
    SetLength(buf1, 1024*8);
    SetLength(buf2, 1024*8);


    whIn1.lpData := PAnsiChar(buf1);

    whIn1.dwBufferLength := Length(buf1);

    whIn1.dwBytesRecorded := 0;

    whIn1.dwUser := 0;

    whIn1.dwFlags := 0;

    whIn1.dwLoops := 0;

    whIn1.lpNext := nil;

    whIn1.reserved := 0;


    whIn2.lpData := PAnsiChar(buf2);

    whIn2.dwBufferLength := Length(buf2);

    whIn2.dwBytesRecorded := 0;

    whIn2.dwUser := 0;

    whIn2.dwFlags := 0;

    whIn2.dwLoops := 0;

    whIn2.lpNext := nil;

    whIn2.reserved := 0;


    waveInPrepareHeader(hWaveIn, @whIn1, SizeOf(TWaveHdr));

    waveInPrepareHeader(hWaveIn, @whIn2, SizeOf(TWaveHdr));
```

```
    waveInAddBuffer(hWaveIn, @whIn1, SizeOf(TWaveHdr));
    waveInAddBuffer(hWaveIn, @whIn2, SizeOf(TWaveHdr));

    waveInStart(hWaveIn);
  end;
end;

//停止录音
procedure TForm1.Button2Click(Sender: TObject);
begin
  waveInStop(hWaveIn);
  waveInUnprepareHeader(hWaveIn, @whIn1, SizeOf(TWaveHdr));
  waveInUnprepareHeader(hWaveIn, @whIn2, SizeOf(TWaveHdr));
  waveInClose(hWaveIn);
end;

//播放录音
procedure TForm1.Button3Click(Sender: TObject);
begin
  whOut.lpData := PAnsiChar(SaveBuf);
  whOut.dwBufferLength := Length(SaveBuf);
  whOut.dwBytesRecorded := 0;
  whOut.dwUser := 0;
  whOut.dwFlags := 0;
  whOut.dwLoops := 1;
  whOut.lpNext := nil;
  whOut.reserved := 0;

  waveOutOpen(@hWaveOut, WAVE_MAPPER, @fmt, Handle, 0, CALLBACK_WINDOW);
  waveOutPrepareHeader(hWaveOut, @whOut, SizeOf(TWaveHdr));
  waveOutWrite(hWaveOut, @whOut, SizeOf(TWaveHdr));
end;

procedure TForm1.WndProc(var m: TMessage);
var
  ordLen: Integer;
begin
  inherited;
  case m.Msg of
    {处理录音消息}
    MM_WIM_OPEN: ;     {此消息只携带了设备句柄}
    MM_WIM_CLOSE: ;    {此消息只携带了设备句柄}
    MM_WIM_DATA: begin {此消息携带了设备句柄和 WaveHdr 指针(LParam)}
```

```
        {保存录制的数据}
        ordLen := Length(SaveBuf);
        SetLength(SaveBuf, ordLen + PWaveHdr(m.LParam).dwBytesRecorded);
         CopyMemory(Ptr(DWORD(SaveBuf)+ordLen), PWaveHdr(m.LParam).lpData,
PWaveHdr(m.LParam).dwBytesRecorded);
        {继续录制}
        waveInAddBuffer(hWaveIn, PWaveHdr(m.LParam), SizeOf(TWaveHdr));
      end;

      {处理播放消息}
      MM_WOM_OPEN: ;      {此消息只携带了设备句柄}
      MM_WOM_CLOSE: ;     {此消息只携带了设备句柄}
      MM_WOM_DONE: begin {此消息携带了设备句柄和 WaveHdr 指针(LParam)}
                     waveOutUnprepareHeader(hWaveOut,    PWaveHdr(m.LParam),
SizeOf(TWaveHdr));
        waveOutClose(hWaveOut);
      end;
  end;
end;


end.
```

使用回调函数处理音频设备发出的消息:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  end;
```

```
var

  Form1: TForm1;


implementation


{$R *.dfm}


uses MMSystem;


var

  whIn1,whIn2, whOut: TWaveHdr;

  hWaveIn,hWaveOut: HWAVE;

  fmt: TWaveFormatEx;

  buf1,buf2,SaveBuf: TBytes;


//回调函数; 容易出错的是: 系统回调函数中不能使用本地变量

procedure WaveProc(hWave: HWAVE; uMsg, dwInstance, dwParam1, dwParam2:

DWORD); stdcall;

var

  ordLen: Integer;

begin

  case uMsg of
```

```
;

    {处理录音消息}
    MM_WIM_OPEN: ;       {此消息只携带了设备句柄}
    MM_WIM_CLOSE: ;      {此消息只携带了设备句柄}
    MM_WIM_DATA: begin {此消息携带了设备句柄和 WaveHdr 指针(dwParam1)}
      {保存录制的数据}
      ordLen := Length(SaveBuf);
      SetLength(SaveBuf, ordLen + PWaveHdr(dwParam1).dwBytesRecorded);
        CopyMemory(Ptr(DWORD(SaveBuf)+ordLen), PWaveHdr(dwParam1).lpData,
PWaveHdr(dwParam1).dwBytesRecorded);
      {继续录制}
      waveInAddBuffer(hWave, PWaveHdr(dwParam1), SizeOf(TWaveHdr));
    end;

    {处理播放消息}
    MM_WOM_OPEN: ;       {此消息只携带了设备句柄}
    MM_WOM_CLOSE: ;      {此消息只携带了设备句柄}
    MM_WOM_DONE: begin {此消息携带了设备句柄和 WaveHdr 指针(dwParam1)}
                      waveOutUnprepareHeader(hWave,     PWaveHdr(dwParam1),
SizeOf(TWaveHdr));
      waveOutClose(hWave);
    end;
  end;
```

```
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := '开始录音';
  Button2.Caption := '停止录音';
  Button3.Caption := '播放录音';
end;


//开始录音
procedure TForm1.Button1Click(Sender: TObject);
begin
  {指定要录制的格式}
  fmt.wFormatTag := WAVE_FORMAT_PCM;
  fmt.nChannels := 2;
  fmt.nSamplesPerSec := 22050;
  fmt.nAvgBytesPerSec := 88200;
  fmt.nBlockAlign := 4;
  fmt.wBitsPerSample := 16;
  fmt.cbSize := 0;

  SaveBuf := nil; {清除已录制的内容}
```

```
    if waveInOpen(@hWaveIn, WAVE_MAPPER, @fmt, DWORD(@WaveProc), 0,
CALLBACK_FUNCTION) = 0 then
  begin
    SetLength(buf1, 1024*8);
    SetLength(buf2, 1024*8);

    whIn1.lpData := PAnsiChar(buf1);
    whIn1.dwBufferLength := Length(buf1);
    whIn1.dwBytesRecorded := 0;
    whIn1.dwUser := 0;
    whIn1.dwFlags := 0;
    whIn1.dwLoops := 0;
    whIn1.lpNext := nil;
    whIn1.reserved := 0;

    whIn2.lpData := PAnsiChar(buf2);
    whIn2.dwBufferLength := Length(buf2);
    whIn2.dwBytesRecorded := 0;
    whIn2.dwUser := 0;
    whIn2.dwFlags := 0;
    whIn2.dwLoops := 0;
    whIn2.lpNext := nil;
    whIn2.reserved := 0;


    waveInPrepareHeader(hWaveIn, @whIn1, SizeOf(TWaveHdr));
    waveInPrepareHeader(hWaveIn, @whIn2, SizeOf(TWaveHdr));
    waveInAddBuffer(hWaveIn, @whIn1, SizeOf(TWaveHdr));
    waveInAddBuffer(hWaveIn, @whIn2, SizeOf(TWaveHdr));


    waveInStart(hWaveIn);
  end;
end;


//停止录音
procedure TForm1.Button2Click(Sender: TObject);
begin
  waveInStop(hWaveIn);
  waveInUnprepareHeader(hWaveIn, @whIn1, SizeOf(TWaveHdr));
  waveInUnprepareHeader(hWaveIn, @whIn2, SizeOf(TWaveHdr));
  waveInClose(hWaveIn);
end;


//播放录音
```

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  whOut.lpData := PAnsiChar(SaveBuf);
  whOut.dwBufferLength := Length(SaveBuf);
  whOut.dwBytesRecorded := 0;
  whOut.dwUser := 0;
  whOut.dwFlags := 0;
  whOut.dwLoops := 1;
  whOut.lpNext := nil;
  whOut.reserved := 0;

    waveOutOpen(@hWaveOut, WAVE_MAPPER, @fmt, DWORD(@WaveProc), 0,
CALLBACK_FUNCTION);
  waveOutPrepareHeader(hWaveOut, @whOut, SizeOf(TWaveHdr));
  waveOutWrite(hWaveOut, @whOut, SizeOf(TWaveHdr));
end;


end.
```

操作 Wave 文件(12): 使用 waveOut...重复播放 wav 文件

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

function GetWaveFmtData(path: string; var fmt: TWaveFormatEx; var buf:
TBytes): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiFmt,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(path), nil, MMIO_READ);
  if hFile = 0 then Exit;

  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));
  ckiRIFF.fccType := mmioStringToFOURCC('WAVE', 0);

  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);

  ckiData.ckid := mmioStringToFOURCC('data', 0);


  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));


  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);


        if (ckiRIFF.ckid = FOURCC_RIFF) and (ckiRIFF.fccType =
mmioStringToFOURCC('WAVE',0)) and

             (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) and

      (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize) and

      (mmioAscend(hFile, @ckiFmt, 0) = MMSYSERR_NOERROR) and

             (mmioDescend(hFile, @ckiData, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) then

  begin

    SetLength(buf, ckiData.cksize);

    Result := (mmioRead(hFile, PAnsiChar(buf), ckiData.cksize) = ckiData.cksize);

  end;
  mmioClose(hFile, 0);
end;

//-------------------------------------------------------------------------------
var
  wh: TWaveHdr;
  hOut: HWAVEOUT;
  fmt: TWaveFormatEx;
  buf: TBytes;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := '打开并播放';
  Button2.Caption := '暂停';
  Button3.Caption := '继续';
end;

procedure WaveProc(hWave: HWAVE; uMsg, dwInstance, dwParam1, dwParam2: DWORD); stdcall;
begin
  case uMsg of
```

```
    MM_WOM_OPEN: ;

    MM_WOM_CLOSE: ;

    MM_WOM_DONE: begin

                      waveOutUnprepareHeader(hWave,    PWaveHdr(dwParam1),
SizeOf(TWaveHdr));

      waveOutClose(hWave);

    end;

  end;

end;


procedure TForm1.Button1Click(Sender: TObject);

const

  path = 'C:\WINDOWS\Media\Windows XP 启动.wav';

begin

  GetWaveFmtData(path, fmt, buf);


  wh.lpData := PAnsiChar(buf);

  wh.dwBufferLength := Length(buf);

  wh.dwBytesRecorded := 0;

  wh.dwUser := 0;

  wh.dwFlags := WHDR_BEGINLOOP or WHDR_ENDLOOP; {关键设置}

  wh.dwLoops := 3;                              {重复播放的次数}
  wh.lpNext := nil;
  wh.reserved := 0;

    waveOutOpen(@hOut,   WAVE_MAPPER,   @fmt,   DWORD(@WaveProc),   0,
CALLBACK_FUNCTION);
  waveOutPrepareHeader(hOut, @wh, SizeOf(TWaveHdr));
  waveOutWrite(hOut, @wh, SizeOf(TWaveHdr));
end;

//暂停
procedure TForm1.Button2Click(Sender: TObject);
begin
  waveOutPause(hOut);
end;

//继续
procedure TForm1.Button3Click(Sender: TObject);
begin
  waveOutRestart(hOut);
end;


end.
```

操作 Wave 文件(13): waveOutGetVolume、waveOutSetVolume

左右声道的音量是单调的; 表示音量的32位整数的低16位是左声道、高16位是右声道.

代码文件:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    ScrollBar1: TScrollBar;
    ScrollBar2: TScrollBar;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure ScrollBar1Change(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses MMSystem;

function GetWaveFmtData(path: string; var fmt: TWaveFormatEx; var buf:
```

```
TBytes): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiFmt,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(path), nil, MMIO_READ);
  if hFile = 0 then Exit;

  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));

  ckiRIFF.fccType := mmioStringToFOURCC('WAVE', 0);
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);
  ckiData.ckid := mmioStringToFOURCC('data', 0);

  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));

  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);

    if (ckiRIFF.ckid = FOURCC_RIFF) and (ckiRIFF.fccType =
mmioStringToFOURCC('WAVE',0)) and
            (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) and
     (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize) and
     (mmioAscend(hFile, @ckiFmt, 0) = MMSYSERR_NOERROR) and
            (mmioDescend(hFile, @ckiData, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) then
  begin
    SetLength(buf, ckiData.cksize);
    Result := (mmioRead(hFile, PAnsiChar(buf), ckiData.cksize) = ckiData.cksize);
  end;

  mmioClose(hFile, 0);
end;

//--------------------------------------------------------------
------------
var
  wh: TWaveHdr;
  hWaveOut: HWAVE;
  fmt: TWaveFormatEx;
  buf: TBytes;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := '打开并播放';
  Button2.Caption := '暂停';
  Button3.Caption := '继续';

  ScrollBar1.Min := 0;
  ScrollBar1.Max := 100;
  ScrollBar2.Min := 0;
  ScrollBar2.Max := 100;

  ScrollBar2.OnChange := ScrollBar1.OnChange;
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
var
  L,R: Word;
begin
  if hWaveOut = 0 then Exit;
  L := Trunc(ScrollBar1.Position / 100 * MAXWORD);
  R := Trunc(ScrollBar2.Position / 100 * MAXWORD);
  waveOutSetVolume(hWaveOut, MakeLong(L, R));
end;


procedure WaveProc(hWave: HWAVE; uMsg, dwInstance, dwParam1, dwParam2:
DWORD); stdcall;
begin
  case uMsg of
      MM_WOM_DONE: waveOutUnprepareHeader(hWave, PWaveHdr(dwParam1),
SizeOf(TWaveHdr));
  end;
end;


procedure TForm1.Button1Click(Sender: TObject);
const
  path = 'C:\WINDOWS\Media\Windows XP 启动.wav';
var
  volume: DWORD;
begin
  GetWaveFmtData(path, fmt, buf);


  wh.lpData := PAnsiChar(buf);
  wh.dwBufferLength := Length(buf);
  wh.dwBytesRecorded := 0;
  wh.dwUser := 0;
  wh.dwFlags := WHDR_BEGINLOOP or WHDR_ENDLOOP;
  wh.dwLoops := 3;
  wh.lpNext := nil;
  wh.reserved := 0;

    waveOutOpen(@hWaveOut, WAVE_MAPPER, @fmt, DWORD(@WaveProc), 0,
CALLBACK_FUNCTION);
  waveOutGetVolume(hWaveOut, @volume);

  ScrollBar1.Position := Trunc(LoWord(volume) / MAXWORD * 100);
  ScrollBar2.Position := Trunc(HiWord(volume) / MAXWORD * 100);

  waveOutPrepareHeader(hWaveOut, @wh, SizeOf(TWaveHdr));
  waveOutWrite(hWaveOut, @wh, SizeOf(TWaveHdr));
end;

//暂停
procedure TForm1.Button2Click(Sender: TObject);
begin
  waveOutPause(hWaveOut);
```

```
end;

//继续
procedure TForm1.Button3Click(Sender: TObject);
begin
  waveOutRestart(hWaveOut);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  if hWaveOut <> 0 then waveOutClose(hWaveOut);
end;

end.
```

窗体文件:

```
object Form1: TForm1
  Left = 0
  Top = 0
```

```
Caption = 'Form1'
ClientHeight = 182
ClientWidth = 342
Color = clBtnFace
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -11
Font.Name = 'Tahoma'
Font.Style = []
OldCreateOrder = False
OnCreate = FormCreate
OnDestroy = FormDestroy
PixelsPerInch = 96
TextHeight = 13
object Button1: TButton
  Left = 32
  Top = 24
  Width = 75
  Height = 25
  Caption = #25171#24320#24182#25773#25918
  TabOrder = 0
  OnClick = Button1Click
```

```
end
object Button2: TButton
  Left = 136
  Top = 24
  Width = 75
  Height = 25
  Caption = #26242#20572
  TabOrder = 1
  OnClick = Button2Click
end
object Button3: TButton
  Left = 240
  Top = 24
  Width = 75
  Height = 25
  Caption = #32487#32493
  TabOrder = 2
  OnClick = Button3Click
end
object ScrollBar1: TScrollBar
  Left = 32
  Top = 80
```

```
    Width = 283
    Height = 17
    PageSize = 0
    TabOrder = 3
    OnChange = ScrollBar1Change
  end
  object ScrollBar2: TScrollBar
    Left = 32
    Top = 120
    Width = 283
    Height = 17
    PageSize = 0
    TabOrder = 4
  end
end
```

操作 Wave 文件(14): waveOutSetPlaybackRate、waveOutSetPitch

这两个参数也都是可以 Get(waveOutGetPlaybackRate、waveOutGetPitch)

设备默认的播放速度是 $00010000, 此值乘以 2 是快一倍, 除以 2 是慢一倍; 最快可到 $000F8000.

设备默认的音高参数是 $00010000, 此值乘以 2 是高一倍, 除以 2 是低一倍; 最高可到 $000F8000.

可能有很多声卡不支持, 我的 IBM 手提就不支持; 不过通过其他技术可以实现的.

代码文件(仅有播放速度的设置代码):

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;

    Button4: TButton;

    TrackBar1: TTrackBar;

    Button5: TButton;

    procedure FormCreate(Sender: TObject);

    procedure Button1Click(Sender: TObject);

    procedure Button2Click(Sender: TObject);

    procedure Button3Click(Sender: TObject);

    procedure Button4Click(Sender: TObject);

    procedure Button5Click(Sender: TObject);

    procedure TrackBar1Change(Sender: TObject);

    procedure FormDestroy(Sender: TObject);

  end;


var

  Form1: TForm1;


implementation


{$R *.dfm}


uses MMSystem;
```

```
function GetWaveFmtData(path: string; var fmt: TWaveFormatEx; var buf:
TBytes): Boolean;
var
  hFile: HMMIO;
  ckiRIFF,ckiFmt,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(path), nil, MMIO_READ);
  if hFile = 0 then Exit;

  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));

  ckiRIFF.fccType := mmioStringToFOURCC('WAVE', 0);
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);
  ckiData.ckid := mmioStringToFOURCC('data', 0);

  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));

  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);
```

```
        if (ckiRIFF.ckid = FOURCC_RIFF) and (ckiRIFF.fccType =
mmioStringToFOURCC('WAVE',0)) and
            (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) and
     (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize) and
     (mmioAscend(hFile, @ckiFmt, 0) = MMSYSERR_NOERROR) and
            (mmioDescend(hFile, @ckiData, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) then
  begin
    SetLength(buf, ckiData.cksize);
    Result := (mmioRead(hFile, PAnsiChar(buf), ckiData.cksize) = ckiData.cksize);
  end;

  mmioClose(hFile, 0);
end;

//---------------------------------------------------------------------------------
-------------
var
  wh: TWaveHdr;
  hWaveOut: HWAVE;
  fmt: TWaveFormatEx;
  buf: TBytes;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.Caption := '打开并播放';
  Button2.Caption := '暂停';
  Button3.Caption := '继续';

  TrackBar1.Min := -4;
  TrackBar1.Max := 4;
  TrackBar1.Position := 0;
end;

procedure WaveProc(hWave: HWAVE; uMsg, dwInstance, dwParam1, dwParam2: DWORD); stdcall;
begin
  case uMsg of
      MM_WOM_DONE: waveOutUnprepareHeader(hWave, PWaveHdr(dwParam1), SizeOf(TWaveHdr));
  end;
end;
```

```
procedure TForm1.Button1Click(Sender: TObject);
const
  path = 'C:\WINDOWS\Media\Windows XP 启动.wav';
var
  volume: DWORD;
begin
  GetWaveFmtData(path, fmt, buf);

  wh.lpData := PAnsiChar(buf);
  wh.dwBufferLength := Length(buf);
  wh.dwBytesRecorded := 0;
  wh.dwUser := 0;
  wh.dwFlags := WHDR_BEGINLOOP or WHDR_ENDLOOP;
  wh.dwLoops := 3;
  wh.lpNext := nil;
  wh.reserved := 0;

  waveOutOpen(@hWaveOut, WAVE_MAPPER, @fmt, DWORD(@WaveProc), 0,
CALLBACK_FUNCTION);
  waveOutPrepareHeader(hWaveOut, @wh, SizeOf(TWaveHdr));
  waveOutWrite(hWaveOut, @wh, SizeOf(TWaveHdr));
```

```
end;

//暂停
procedure TForm1.Button2Click(Sender: TObject);
begin
  waveOutPause(hWaveOut);
end;

//继续
procedure TForm1.Button3Click(Sender: TObject);
begin
  waveOutRestart(hWaveOut);
end;

//调整播放速度
procedure TForm1.TrackBar1Change(Sender: TObject);
const
  mid = $00010000;
var
  pos, rate: Integer;
begin
  pos := TTrackBar(Sender).Position;
  if pos > 0 then
    rate := mid shl pos
  else
    rate := mid shr Abs(pos);

  waveOutSetPlaybackRate(hWaveOut, rate);

  Text := IntToStr(pos);
end;

//判断设备是否支持播放速度调整
procedure TForm1.Button4Click(Sender: TObject);
var
  waveOutCaps: TWaveOutCaps;
begin
  waveOutGetDevCaps(WAVE_MAPPER, @waveOutCaps, SizeOf(TWaveOutCaps));
  if waveOutCaps.dwSupport and WAVECAPS_PLAYBACKRATE = WAVECAPS_PLAYBACKRATE then
    ShowMessage('默认设备支持播放速度调整.')
  else
    ShowMessage('默认设备不支持播放速度调整!');
```

```
end;

//判断设备是否支持声调变化
procedure TForm1.Button5Click(Sender: TObject);
var
  waveOutCaps: TWaveOutCaps;
begin
  waveOutGetDevCaps(WAVE_MAPPER, @waveOutCaps, SizeOf(TWaveOutCaps));
  if waveOutCaps.dwSupport and WAVECAPS_PITCH = WAVECAPS_PITCH then
    ShowMessage('默认设备支持声调变化.')
  else
    ShowMessage('默认设备不支持声调变化!');
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  if hWaveOut <> 0 then waveOutClose(hWaveOut);
end;

end.
```

窗体文件:

```
object Form1: TForm1
  Left = 0
  Top = 0
  Caption = 'Form1'
  ClientHeight = 204
  ClientWidth = 342
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'Tahoma'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  OnDestroy = FormDestroy
  PixelsPerInch = 96
  TextHeight = 13
  object Button1: TButton
    Left = 32
    Top = 24
    Width = 75
    Height = 25
    Caption = #25171#24320#24182#25773#25918
    TabOrder = 0
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 136
    Top = 24
    Width = 75
    Height = 25
    Caption = #26242#20572
    TabOrder = 1
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 240
    Top = 24
    Width = 75
    Height = 25
    Caption = #32487#32493
    TabOrder = 2
```

```
    OnClick = Button3Click
  end
  object TrackBar1: TTrackBar
    Left = 32
    Top = 88
    Width = 283
    Height = 45
    TabOrder = 3
    OnChange = TrackBar1Change
  end
  object Button4: TButton
    Left = 152
    Top = 139
    Width = 163
    Height = 25
                                                        Caption                 =
#21028#26029#35774#22791#26159#21542#25903#25345#36895#24230#3
5843#25972
    TabOrder = 4
    OnClick = Button4Click
  end
  object Button5: TButton
```

```
    Left = 152
    Top = 170
    Width = 163
    Height = 25
    Caption =
#21028#26029#35774#22791#26159#21542#25903#25345#22768#35843#2
1464#21270
    TabOrder = 5
    OnClick = Button5Click
  end
end
```

操作 Wave 文件(15): 合并与剪裁 wav 文件

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
```

```
type

  TForm1 = class(TForm)

    Button1: TButton;

    Button2: TButton;

    procedure Button1Click(Sender: TObject);

    procedure Button2Click(Sender: TObject);

  end;


var

  Form1: TForm1;


implementation


{$R *.dfm}


uses MMSystem;


//从指定 wav 文件中获取格式信息和波形数据的函数

function GetWaveFmtData(const path: string; var fmt: TWaveFormatEx; var buf:

TBytes): Boolean;

var
```

```
  hFile: HMMIO;
  ckiRIFF,ckiFmt,ckiData: TMMCKInfo;
begin
  Result := False;
  hFile := mmioOpen(PChar(path), nil, MMIO_READ);
  if hFile = 0 then Exit;
  ZeroMemory(@ckiRIFF, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));
  ckiRIFF.fccType := mmioStringToFOURCC('WAVE', 0);
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);
  ckiData.ckid := mmioStringToFOURCC('data', 0);
  ZeroMemory(@fmt, SizeOf(TWaveFormatEx));
  mmioDescend(hFile, @ckiRIFF, nil, MMIO_FINDRIFF);
      if (ckiRIFF.ckid = FOURCC_RIFF) and (ckiRIFF.fccType =
mmioStringToFOURCC('WAVE',0)) and
          (mmioDescend(hFile, @ckiFmt, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) and
     (mmioRead(hFile, @fmt, ckiFmt.cksize) = ckiFmt.cksize) and
     (mmioAscend(hFile, @ckiFmt, 0) = MMSYSERR_NOERROR) and
          (mmioDescend(hFile, @ckiData, @ckiRIFF, MMIO_FINDCHUNK) =
MMSYSERR_NOERROR) then
```

```
  begin
    SetLength(buf, ckiData.cksize);
    Result := (mmioRead(hFile, PAnsiChar(buf), ckiData.cksize) = ckiData.cksize);
  end;
  mmioClose(hFile, 0);
end;

//根据格式信息和波形数据建立 wav 文件的函数
function CreateWave(const path: string; const fmt: TWaveFormatEx; const buf:
TBytes): Boolean;
var
  h: HMMIO;
  ckiRiff, ckiFmt, ckiData: TMMCKInfo;
begin
  ZeroMemory(@ckiRiff, SizeOf(TMMCKInfo));
  ckiRiff.cksize := 44 - 8 + Length(buf);
  ckiRiff.fccType := mmioStringToFOURCC('WAVE', 0);

  ZeroMemory(@ckiFmt, SizeOf(TMMCKInfo));
  ckiFmt.ckid := mmioStringToFOURCC('fmt', 0);

  ZeroMemory(@ckiData, SizeOf(TMMCKInfo));
  ckiData.ckid := mmioStringToFOURCC('data', 0);
  ckiData.cksize := Length(buf);

  h := mmioOpen(PChar(path), nil, MMIO_CREATE or MMIO_WRITE);
    if (h <> 0) and (mmioCreateChunk(h, @ckiRiff, MMIO_CREATERIFF) = MMSYSERR_NOERROR) and
    (mmioCreateChunk(h, @ckiFmt, 0) = MMSYSERR_NOERROR) and
             (mmioWrite(h, PAnsiChar(@fmt), SizeOf(TPCMWaveFormat)) = SizeOf(TPCMWaveFormat)) and
    (mmioAscend(h, @ckiFmt, 0) = MMSYSERR_NOERROR) and
    (mmioCreateChunk(h, @ckiData, 0) = MMSYSERR_NOERROR) then
    Result := (mmioWrite(h, PAnsiChar(buf), Length(buf)) = Length(buf));
  mmioClose(h, 0);
end;


//截取 wav 文件，本例截留了文件的 1/4
procedure TForm1.Button1Click(Sender: TObject);
const
  pathSource = 'C:\WINDOWS\Media\Windows XP 启动.wav';
  pathDest = 'C:\Temp\New1.wav';
var
```

```
  fmt: TWaveFormatEx;
  buf: TBytes;
begin
  GetWaveFmtData(pathSource, fmt, buf);
  SetLength(buf, Length(buf) div 4);
  CreateWave(pathDest, fmt, buf);
end;


//合并 wav 文件
procedure TForm1.Button2Click(Sender: TObject);
const
  path1 = 'C:\WINDOWS\Media\Windows XP 启动.wav';
  path2 = 'C:\WINDOWS\Media\Windows XP 关机.wav';
  pathDest = 'C:\Temp\New2.wav';
var
  fmt1,fmt2: TWaveFormatEx;
  buf1,buf2: TBytes;
  oldLen: Integer;
begin
  GetWaveFmtData(path1, fmt1, buf1);
  GetWaveFmtData(path2, fmt2, buf2);
  if CompareMem(@fmt1, @fmt2, SizeOf(TWaveFormatEx)) then
  begin
    oldLen := Length(buf1);
    SetLength(buf1, Length(buf1) + Length(buf2));
    CopyMemory(@buf1[oldLen], Pointer(buf2), Length(buf2));
    CreateWave(pathDest, fmt1, buf1);
  end else ShowMessage('文件格式不一致，没有执行合并!');
end;


end.
```