# ゲームプログラミング

## ▶　DirectX Audio－第７回　Waveファイルの読み込み　◀

DirectSoundは、Waveファイルをサウンドバッファに読み込む方法を提供していません。また、Waveファイルを読み込むAPIも用意されていないので、解析しながら読み込む必要があります。

### Waveファイルの構成

サウンドバッファが保持するサウンドデータの形式は、通常のWaveファイルのイメージ(PCM形式)とまったく同じなので、Waveファイルのイメージをそのまま転送するだけで、DirectSoundで制御できるようになります。しかし、DirectSoundには、Waveファイルを読み込むメソッドや関数はありません。Waveファイルを解析しながら読み込まなければなりません。そのためには、Waveファイルの構造を正しく理解しておく必要があります。

Waveファイルは、AVIファイルなどでも使われているRIFF(Resource Interchange File Format)形式になっています。一般的なWaveファイルは、以下のような構成になっています。

| |
|---|
| RIFFの４文字('RIFF'ファイルのヘッダ) |
| この部分以降のデータサイズ ※１ |
| WAVEの４文字('WAVE'チャンクのヘッダ) |
| fmt□の４文字('fmt 'チャンクのヘッダ)<br>DWORD　fmt領域(正式名称チャンク)のサイズ<br>WORD　フォーマットID<br>WORD　チャネル数<br>DWORD　サンプリング周波数<br>DWORD　平均データ速度<br>WORD　ブロックサイズ<br>WORD　１サンプル当たりのビット数<br>WORD　ヘッダ拡張部サイズ(byte) ※２<br>～　ヘッダ拡張部 ※２ ※３ |
| factの４文字('fact'チャンクのヘッダ)<br>DWORD　fact領域のサイズ<br>DWORD　全サンプル数 |
| dataの４文字('data'チャンクのヘッダ)<br>DWORD　data領域のサイズ<br>～　WAVEファイルのイメージ |

※１　このサイズはWAVEヘッダ(４バイト)、fmt領域、fact領域、data領域の合計です。
通常は、「ファイルサイズ－８」です。

※２　フォーマットIDが１(リニアPCM)のときは存在しません。

※３　ヘッダ拡張部サイズが０のときは存在しません

※４　fmt領域、fact領域、data領域はWAVEチャンクのサブチャンクです。

※５　fmt領域、fact領域、data領域は順不同です。どのような順番で存在してもかまいません。

※６　DirectSoundがサポートしているのはフォーマットIDが１のみです。他の形式はACMでリニアPCM形式にデコードする必要があります。

※７　フォーマットIDが１のとき、fmt領域は、fmt領域のサイズを除けば、WAVEFORMATEX構造体からcbSizeメンバを除いたものとまったく同じ構造です。

○フォーマットID

| | | |
|---|---|---|
| 0x0001　リニアPCM | 0x0011　ADPCM(IMA/DVI) | 0x0022　TrueSpeech |
| 0x0002　MS ADPCM | 0x0012　MediaSpace ADPCM | 0x0030　DOLBY AC2 |
| 0x0005　IBM CSVD | 0x0013　Sierra ADPCM | 0x0031　GSM 6.10 |
| 0x0006　A-Law | 0x0014　ADPCM(G.723) | 0x0040　ADPCM(G.721) |
| 0x0007　μ-Law | 0x001E　MPEG1 Layer-3 | 0x0200　CREATIVE ADPCM |
| 0x0010　OKI ADPCM | 0x0020　YAMAHA ADPCM | 0x0300　FM-TOWNS SND |

(この他にも多数存在します)

○リニアPCMについて

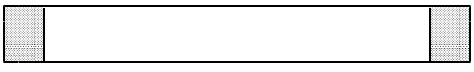
リニアPCM(圧縮されていないPCM)では、１サンプルあたりのビット数は、８と16が存在します。８ビットの場合は0～255でサウンドデータを表し、128(0x80)が無音状態です。16ビットの場合は-32,768～32,767でサウンドデータを表し、０が無音状態です。ステレオデータは、左チャネル、右チャネルの順で格納されています。また、ヘッダ拡張部やfact領域がないものが多く存在します。

## サウンドバッファのロックとリングバッファ

Waveファイルをサウンドバッファに転送するには、サウンドバッファが格納されている領域のアドレス(ポインタ)を取得し、メモリ操作関数やファイル操作関数、マルチメディア入出力関数などで直接的に転送します。

サウンドバッファのポインタを取得するには、DirectSoundBufferオブジェクトのLockメソッドを使います。このメソッドは、２つの領域のポインタを返すことがあります。それは、サウンドバッファがリングバッファになっているためです。

リングバッファ(循環バッファ)とは､概念的にバッファの最後と先頭がつながっているということで､バッファの最後尾の次は先頭に戻ります。たとえば、1000バイトの大きさを持つバッファの先頭から800バイト目を基点に400バイトをロックしたとします。通常のバッファなら、領域の大きさを超えるサイズをロックすることはできませんが､リングバッファである場合はロックすることができ､「800～999」と「0～199」の２つの領域がロックされます。




▲リングバッファが循環する例

Lockメソッドでロックされた領域は、書き込むための領域です。この領域から読み取ることは許可されていません。バッファがサウンドカード上のメモリに配置されている場合、返されたポインタがシステムメモリにコピーされた一時バッファのアドレスとなる可能性があるからです。この領域からデータを読み込んでもまったく意味がありません。ロックを解除すると一時バッファのデータがサウンドカードに転送されます。

書き込みが終わったら、ただちにロックを解除する必要があります。ロックを解除しないと、環境によっては、音声の分断や無音状態が発生する場合があります。ロックの解除は、DirectSoundBufferオブジェクトののUnlockメソッドで行います。

### Lockメソッド

－説明－

Lockメソッドは、サウンドバッファの全部または一部をデータ書き込み用に準備し、データを書き込むためのポインタを返します。

－パラメータ－

１つ目の引数は、バッファの先頭からロック開始位置までのバイト数(オフセット)です。７つ目の引数がDSBLOCK_FROMWRITECURSORフラグの場合は無視されます。

２つ目の引数は、ロックするバイト数です。７つ目の引数がDSBLOCK_ENTIREBUFFERフラグの場合は無視されます。

３つ目の引数は、バッファの最初にロックされた領域へのポインタを受け取る変数のアドレスです。

４つ目の引数は、最初にロックされた領域のバイト数を受け取る変数のアドレスです。この値が２つ目の引数より小さい場合、ロックは循環され、２番目にロックされた領域があることを表します。

５つ目の引数は、２番目にロックされた領域へのポインタを受け取る変数のアドレスです。循環しない場合、変数の内容がNULLになります。

６つ目の引数は、２番目にロックされた領域のバイト数を受け取る変数のアドレスです。循環しない場合、変数の値が０になります。

７つ目の引数は、ロックの動作を指定するフラグです。以下のフラグが定義されています。

| | |
|---|---|
| DSBLOCK_ENTIREBUFFER | バッファ全体をロックします |
| DSBLOCK_FROMWRITECURSOR | 書き込みカーソルの位置からロックします |

－戻り値－

成功した場合はDS_OK、それ以外はエラーの原因をエラーコードで返します。

### Unlockメソッド

－説明－

Unlockメソッドは、サウンドバッファのロックを解除します。ロックを解除すると、書き込まれたデータが有効になり、書き込みカーソルの位置が更新されます。

－パラメーター－
　１つ目の引数は、Lockメソッドで取得した最初の領域のアドレスです。
　２つ目の引数は、最初の領域に実際に書き込んだバイト数です。
　３つ目の引数は、Lockメソッドで取得した２番目にロックされた領域のアドレスです。
　４つ目の引数は、２番目の領域に実際に書き込んだバイト数です。

－戻り値－
　成功した場合はDS_OK、それ以外はエラーの原因をエラーコードで返します。

## マルチメディア入出力関数

　RIFF形式のファイルは、マルチメディア入出力関数(mmio関数)を使うと簡単に読み込むことができます。マルチメディア入出力関数では、RIFF形式のファイルを領域(チャンク)単位で扱うことができます。Waveファイルの読み込みでは、以下の関数を使用します。

○mmioOpen(LPSTR szFilename, LPMMIOINFO lpmmioinfo, DWORD dwOpenFlags)
　入出力をバッファリングしてファイルを開きます。この関数が返すハンドルは、標準のファイルハンドル(HANDLE)ではないため、マルチメディアファイル入出力関数以外のファイル操作関数では使用できません。

－パラメーター－
　LPSTR　　szFilename…ファイル名です。最後のNULLを含めて128バイトを越えることはできません。
　LPMMIOINFO lpmmioinfo…追加情報が入ったMMIOINFO構造体のアドレスです。
　DWORD　　dwOpenFlags…オープン操作のためのフラグを指定します。

－戻り値－
　関数が成功すると、開いたファイルのハンドル(HMMIO型)が返ります。ファイルを開くことができない場合は、NULLが返ります。

○mmioFOURCC(CHAR ch0, CHAR ch1, CHAR ch2, CHAR ch3)
　mmioFOURCC関数の実態はマクロです。４つの文字コードを４文字コードに変換します。処理内容は以下のようになっています。

```
((DWORD)(BYTE)(ch0)       | ((DWORD)(BYTE)(ch1) << 8) |
((DWORD)(BYTE)(ch2) << 16) | ((DWORD)(BYTE)(ch3) << 24 ))
```

－パラメーター－
　CHAR ch0～ch3…４つの文字コードです。

－戻り値－
　４つの文字コードから変換された、４文字コードを返します。

○mmioDescend(HMMIO hmmio, LPMMCKINFO lpck, LPMMCKINFO lpckParent, UINT wFlags)
　mmioOpen関数で開いたファイルのチャンク(領域)に進入します。また、指定されたチャンクを検索することもできます。

－パラメーター－
　HMMIO　　hmmio…RIFFファイルのハンドルを指定します。
　LPMMCKINFO lpck…MMCKINFO構造体のアドレスを指定します。
　LPMMCKINFO lpckParent…検索するチャンクの親を識別するMMCKINFO構造体のアドレスを指定します。
　　このパラメータがNULLでない場合、mmioDescend関数が呼び出されて親チャンクに進入すると、mmioDescend関数は参照するMMCKINFO構造体に値が設定されていると想定して、親チャンク内でチャンクを検索します。親チャンクが指定されていない場合、このパラメータはNULLに設定します。
　UINT　　wFlags…検索フラグを指定します。フラグが指定されていない場合、mmioDescend関数は現在のファイル位置のチャンクの先頭に進入します。
　　MMIO_FINDCHUNK　指定されたチャンク識別子のチャンクを検索します。
　　MMIO_FINDRIFF　親チャンクが"RIFF"で、指定されたサブチャンクを検索します。

－戻り値－
　関数が成功するとMMSYSERR_NOERRORが返ります。失敗するとMMIOERR_CHUNKNOTFOUNDが返ります。

○mmioRead(HMMIO hmmio, HPSTR pch, LONG cch)
　mmioOpen関数で開いたファイルから、指定されたバイト数を読み取ります。

－パラメーター－
HMMIO hmmio…読み取るファイルのハンドルを指定します。
HPSTR pch…ファイルから読み取られたデータを格納するアドレスを指定します。
LONG cch…ファイルから読み取るバイト数を指定します。

－戻り値－
関数が成功すると、実際に読み取られたバイト数が返ります。ファイルの終わりに到達し、それ以上読み取ることができない場合は、０が返ります。ファイルの読み取りエラーが発生した場合は－１が返ります。

○mmioAscend(HMMIO hmmio, LPMMCKINFO lpck, UINT wFlags)
mmioDescend関数で進入したチャンクから退出します。

－パラメーター－
HMMIO hmmio…RIFFファイルのハンドルを指定します。
LPMMCKINFO lpck…mmioDescend関数で使用したMMCKINFO構造体のアドレスを指定します。
UINT wFlags…予約されています。０を指定してください。

－戻り値－
関数が成功するとMMSYSERR_NOERRORが返ります。それ以外はエラーコードが返ります。

○mmioClose(HMMIO hmmio, UINT wFlags)
mmioOpen関数で開いたファイルを閉じます。

－パラメーター－
HMMIO hmmio…閉じるファイルのハンドルを指定します。
UINT wFlags…クローズ操作のためのフラグを指定します。HMMIO型以外のファイルハンドルを指定して開かれたファイルの場合のみ使用します。

－戻り値－
関数が成功すると、０が返ります。

マルチメディア入出力関数を使ったソースファイルをビルドする場合は、ヘッダファイル<mmsystem.h>(または<dmusici.h>)とライブラリ"winmm.lib"が必要になります。

## ○ 課題 ○

以下のプログラムは、Waveファイルからサウンドバッファを生成するDXACreateBufferFromFile関数です。関数の仕様をよく読み、プログラムを完成させましょう。

DXACreateBufferFromFile関数
－説明－
DXACreateBufferFromFile関数は、Waveファイルからサウンドバッファを生成します。生成されたサウンドバッファは、Waveファイルとまったく同じフォーマットを持ちます。

－パラメーター－
const DXA_SB dxaBuf…セカンダリバッファの指定。DXA_SB列挙体を指定
LPSTR lpszFileName…読み込むファイル名

－戻り値－
成功した場合はtrue、それ以外はfalseを返す。

```
/*****************************************************************************************************/
/*                                    セカンダリバッファ生成(ファイル)                                 */
/*****************************************************************************************************/
bool DXACreateBufferFromFile(const DXA_SB dxaBuf, LPSTR lpszFileName)
{
    if(NULL == g_lpDSound8) {
        OutputDebugString("*** Error - DirectSound未初期化(DXACreateBufferFromFile)\n");
        return false;
    }

    DXAReleaseBuffer(dxaBuf);

    // ファイルオープン
    HMMIO   hmioFile = ????????(lpszFileName, NULL, MMIO_READ | MMIO_ALLOCBUF);
```

```
if(NULL == hmioFile) {
    OutputDebugString("*** Error - ファイルオープン失敗(DXACreateBufferFromFile)\n");
    return false;
}

MMCKINFO   mmciParent, mmciChild;
ZeroMemory(&mmciParent, sizeof(mmciParent));
ZeroMemory(&mmciChild,  sizeof(mmciChild ));

// 'WAVE'チャンクへ進入
mmciParent.fccType = mmioFOURCC('W', 'A', 'V', 'E');
if(MMSYSERR_NOERROR != mmioDescend(hmioFile, &mmciParent, NULL, MMIO_FINDRIFF)) {
    OutputDebugString("*** Error - 'WAVE'チャンクが見つかりません(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    return false;
}

// 'WAVE'チャンクの'fmt 'チャンクへ進入
mmciChild.ckid = mmioFOURCC('f', 'm', 't', ' ');
if(MMSYSERR_NOERROR != mmioDescend(hmioFile, &mmciChild, &mmciParent, MMIO_FINDCHUNK)) {
    OutputDebugString("*** Error - 'fmt 'チャンクが見つかりません(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    return false;
}

// フォーマット情報取得
WAVEFORMATEX   wfxFormat;
if(sizeof(wfxFormat) != (DWORD)mmioRead(hmioFile, (HPSTR)&wfxFormat, sizeof(wfxFormat))) {
    OutputDebugString("*** Error - フォーマット情報取得失敗(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    return false;
}

// 'fmt 'チャンク退出
if(MMSYSERR_NOERROR != mmioAscend(hmioFile, &mmciChild, 0)) {
    OutputDebugString("*** Error - 'fmt 'チャンク退出失敗(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    return false;
}

// 'WAVE'チャンクの'data'チャンクへ進入
mmciChild.ckid = mmioFOURCC('d', 'a', 't', 'a');
if(MMSYSERR_NOERROR != mmioDescend(hmioFile, &mmciChild, &mmciParent, MMIO_FINDCHUNK)) {
    OutputDebugString("*** Error - 'data'チャンクが見つかりません(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    return false;
}

// セカンダリバッファ生成
if(false == ????????????????(dxaBuf, &wfxFormat, mmciChild.cksize)) {
    OutputDebugString("*** Error - セカンダリバッファ生成失敗(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    return false;
}

// バッファロック
LPVOID    lpDSBMem [2];
DWORD     dwDSBSize[2];
if(DS_OK != g_lpDSBuf8[dxaBuf]->????(0, 0, &lpDSBMem[0], &dwDSBSize[0], &lpDSBMem[1], &dwDSBSize[1],
                                     DSBLOCK_???????????)) {
    OutputDebugString("*** Error - バッファロック失敗(DXACreateBufferFromFile)\n");
    mmioClose(hmioFile, 0);
    DXAReleaseBuffer(dxaBuf);
    return false;
}

// イメージ読み込み
const LONG   lActual = ????????(hmioFile, (HPSTR)lpDSBMem[0], dwDSBSize[0]);
g_lpDSBuf8[dxaBuf]->Unlock(lpDSBMem[0], lActual != -1 ? lActual : 0, lpDSBMem[1], 0);
mmioClose(hmioFile, 0);
if(dwDSBSize[0] != (DWORD)lActual) {
    OutputDebugString("*** Error - イメージ読み込み失敗(DXACreateBufferFromFile)\n");
    DXAReleaseBuffer(dxaBuf);
    return false;
```

```
    }
    return true;
}
```