目录

[MIDIキーボード入力アプリができた « モルタルコのプログラマ日記 1](#_Toc501439002)

[Windows APIによるMIDIプログラミング 3](#_Toc501439003)

[midiInOpen 関数 3](#_Toc501439004)

[MMRESULT 3](#_Toc501439005)

[midiInClose 関数 3](#_Toc501439006)

[midiInGetNumDevs 関数 3](#_Toc501439007)

[midiInGetDevCaps 関数 3](#_Toc501439008)

[http://cpansearch.perl.org/src/NI-S/Audio-1.029/Play/config/mmsystem.h 3](#_Toc501439009)

[MIDIINCAPS structure (Windows) 3](#_Toc501439010)

[midiInStart 関数 3](#_Toc501439011)

[midiInStop 関数 3](#_Toc501439012)

[midiInGetErrorText 関数 3](#_Toc501439013)

[＠IT：.NET TIPS Win32 APIやDLL関数を呼び出すには？ - C# 3](#_Toc501439014)

[意外と知られていない Windows のエラーコードの数々 - 岩田 雅樹 のブログ - Site Home - MSDN Blogs 4](#_Toc501439015)

[MIDI Table 4](#_Toc501439016)

[README.mdファイル。マークダウン記法まとめ | codechord 4](#_Toc501439017)

[白執事の徒然なる日々 VB.NET で USBデバイス を検出する！ 4](#_Toc501439018)

# MIDIキーボード入力アプリができた « モルタルコのプログラマ日記

2010/09/18 - MIDIキーボード入力アプリができた

C#でMIDIキーボード入力をやってみる。midiInOpenにコールバックを登録しておけば，midiInStart/midiInStopでメッセージを受け取れる。

{

[DllImport("winmm.dll")]

public static extern uint midiOutOpen(

ref IntPtr phmo,

uint uDeviceID,

IntPtr dwCallback,

IntPtr dwInstance,

uint fdwOpen);

public delegate void CallBackDelegate(

IntPtr hdrvr,

uint uMsg,

uint dwUser,

uint dw1,

uint dw2);

}

public class MIDIIn

{

private Win32MIDI.CallBackDelegate \_callback = null;

public void Open()

{

\_callback = new Win32MIDI.CallBackDelegate(CallBack);

Win32MIDI.midiInOpen(

ref \_midiin,

\_mididevice,

(IntPtr)Marshal.GetFunctionPointerForDelegate(\_callback),

(IntPtr)0,

Win32MIDI.CALLBACK\_FUNCTION);

}

private void CallBack(IntPtr hdrvr, uint uMsg, uint dwUser, uint dw1, uint dw2)

{

int state = (int)dw1 & 0xff;

int data1 = (int)dw1 >> 8 & 0xff;

int data2 = (int)dw1 >> 16 & 0xff;

}

}

最初，C++で実装するときと同じようにCallBackをstaticメソッドにしていたのだが，インスタンスのメンバ変数を使いたくて，midiInOpenのdwInstanceにthisを渡すことを考えていた。でもthisをIntPtrに変換する方法というか変換していいのかを調べても何も情報は見つからない。最終的に「GetFunctionPointerForDelegateはthisポインタをうまいこと扱ってくれる」というような記述を見つけて，思いきってstaticを取ったらそのまま動いた。C++の発想だとstaticにしないと動かないのでメンバ関数をそのまま渡すというのは思いつきもしなかった。こんな感じで半日を無駄にしてしまった。

その後は順調に進んで，キーボード練習ソフトっぽいものはできあがった。そろそろもばもば（？）に投げてみよう。

# Windows APIによるMIDIプログラミング

## midiInOpen 関数

midiInOpen

<https://msdn.microsoft.com/ja-jp/library/ms712085.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712073.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712209.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712205.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712210.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712204.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712207.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712206.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712208.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712089.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712070.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712073.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712079.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712076.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712082.aspx>

<https://msdn.microsoft.com/ja-jp/library/ms712085.aspx>

指定された MIDI 入力デバイスをオープンします。

MMRESULT midiInOpen(

LPHMIDIIN lphMidiIn,

UINT uDeviceID,

DWORD dwCallback,

DWORD dwCallbackInstance,

DWORD dwFlags

);

パラメータ

*lphMidiIn*

**HMIDIIN**ハンドルのアドレスを指定します。この場所には、オープンした MIDI 入力デバイスを識別するハンドルが入ります。ハンドルは、ほかの MIDI 入力関数の呼び出しでこのデバイスを識別するために使われます。

*uDeviceID*

オープンする MIDI 入力デバイスの識別子を指定します。

*dwCallback*

着信 MIDI メッセージに関する情報によって呼び出されるコールバック関数、スレッド識別子、またはウィンドウのハンドルのアドレスを指定します。コールバック関数の詳細については、**MidiInProc**を参照してください。

*dwCallbackInstance*

コールバック関数に渡されるユーザーインスタンスデータを指定します。このパラメータは、ウィンドウコールバック関数またはスレッドとともには使われません。

*dwFlags*

デバイスをオープンするためのコールバックフラグと、高速データ転送を調節するステータスフラグ(任意）を指定します。次の値のいずれかになります。

*CALLBACK\_FUNCTION*

*dwCallback*パラメータはコールバックプロシージャアドレスです。

*CALLBACK\_NULL*

コールバック機構はありません。この値が既定の設定です。

*CALLBACK\_THREAD*

*dwCallback*パラメータはスレッド識別子です。

*CALLBACK\_WINDOW*

*dwCallback*パラメータはウィンドウハンドルです。

*MIDI\_IO\_STATUS*

このパラメータに CALLBACK\_FUNCTION も指定する場合は、MIM\_DATA メッセージとともに MIM\_MOREDATA メッセージがコールバック関数に送信されます。このパラメータに CALLBACK\_WINDOW も指定する場合は、MM\_MIM\_DATA メッセージとともに MM\_MIM\_MOREDATA メッセージがウィンドウに送信されます。このフラグはイベントコールバックまたはスレッドコールバックには影響しません。

コールバック機構を使うほとんどのアプリケーションでは、このパラメータに CALLBACK\_FUNCTION を指定します。

戻り値

関数が成功すると、MMSYSERR\_NOERROR が返ります。関数が失敗すると、エラーが返ります。返されるエラー値は次のとおりです。

|  |  |
| --- | --- |
| **値** | **説明** |
| MMSYSERR\_ALLOCATED | 指定されたリソースは既に割り当てられています。 |
| MMSYSERR\_BADDEVICEID | 指定されたデバイス識別子は範囲外です。 |
| MMSYSERR\_INVALFLAG | *dwFlags*パラメータで指定されたフラグは無効です。 |
| MMSYSERR\_INVALPARAM | 指定されたポインタまたは構造体は無効です。 |
| MMSYSERR\_NOMEM | システムはメモリを割り当てられないか、またはロックできません。 |

解説

システムに存在する MIDI 入力デバイスの数を調べるには、**midiInGetNumDevs**関数を使います。*wDeviceID* パラメータで指定するデバイス識別子の値は、0 から存在するデバイス数未満(存在するデバイス数から 1 を引いた数）までの範囲です。  
コールバック情報を受け取るウィンドウまたはスレッドが指定されると、MM\_MIM\_OPEN、MM\_MIM\_CLOSE、MM\_MIM\_DATA、MM\_MIM\_LONGDATA、MM\_MIM\_ERROR、MM\_MIM\_LONGERROR および MM\_MIM\_MOREDATA メッセージがウィンドウプロシージャまたはスレッドに送信され、MIDI 入力の進捗状況が示されます。  
コールバック情報を受け取る関数が指定されると、MIM\_OPEN、MIM\_CLOSE、MIM\_DATA、MIM\_LONGDATA、MIM\_ERROR、MIM\_LONGERROR および MIM\_MOREDATA メッセージがその関数に送信され、MIDI 入力の進捗状況が示されます。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用

参照

、、、、[midiInGetNumDevs](https://msdn.microsoft.com/ja-jp/library/cc410520.aspx)、[MidiInProc](https://msdn.microsoft.com/ja-jp/library/cc410529.aspx)、、、、、、、、、、、、

### MMRESULT

MMRESULT

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gifMMRESULTについて

MMRESULTは、マルチメディア関連のAPIで用いられる変数の型で、関数が戻すリターンコードを示します。その型は、「mmsystem.h」の中で以下のように定義されています。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

typedef UINT MMRESULT; /\* error return code, 0 means no error \*/

ゼロは特別な値で、エラーが無いことを示します。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

#define MMSYSERR\_NOERROR 0 /\* no error \*/

MMRESULTは、いくつかのグループに分類されます。エラーコードはグループを作っており、その先頭の値が次のように決められています。一般的なエラーは、「MMSYSERR\_BASE」を先頭に配置され、例えば、MIDI関連のエラーは、「MIDIERR\_BASE」を先頭に配置される、という要領です。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

#define MMSYSERR\_BASE 0

#define WAVERR\_BASE 32

#define MIDIERR\_BASE 64

#define TIMERR\_BASE 96

#define JOYERR\_BASE 160

#define MCIERR\_BASE 256

#define MIXERR\_BASE 1024

ここでサンプルとしてWAV関連の定義を示します。「WAVERR\_BASE」から始まる値になっていることを確認してください。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

#define WAVERR\_BADFORMAT (WAVERR\_BASE + 0) /\* unsupported wave format \*/

#define WAVERR\_STILLPLAYING (WAVERR\_BASE + 1) /\* still something playing \*/

#define WAVERR\_UNPREPARED (WAVERR\_BASE + 2) /\* header not prepared \*/

#define WAVERR\_SYNC (WAVERR\_BASE + 3) /\* device is synchronous \*/

#define WAVERR\_LASTERROR (WAVERR\_BASE + 3) /\* last error in range \*/

MMRESULTを戻す関数は、各＊＊＊\_BASEで定義されている名前から分かるように、以下のグループが定義されています。

* ウェーブ関連
* MIDI関連
* タイマー関連
* ジョイスティック関連
* MCIコマンド関連
* ミキサー関連

特にウェーブやMIDIなど、サウンドを扱う上で重要な関数群が含まれています。

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gifアンマネージ環境

アンマネージ環境（C/C++）でMMRESULTを利用するには、「stdafx.h」などに「mmsystem.h」をインクルードすれば良く、特に問題となる点はありません。

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gifマネージ環境

一方、マネージ環境（C#）からウェーブやMIDI関連のWin32APIを使うためには、APIの引数や戻り値などを定義した宣言をソースに含めなければなりません。当然、MMRESULTも定義することになります。

|  |  |  |
| --- | --- | --- |
| http://www.cactussoft.co.jp/Sarbo/res/imgMemo.gif | |  | | --- | | 本来であれば、マイクロソフトがマネージ環境でウェーブやMIDI関連のWin32APIを使えるよう、定義ファイルやラッパークラスを提供することが望ましいのですが、そのようなものは今の所、存在しません。マネージ環境からWin32APIを呼び出す手続きについては、「プラットフォーム呼び出し」、「マーシャリング」などのセクションで解説しているので参照してください。 | |

C#には「typedef」に相当する機能が無いため、MMRESULTをuint型として定義することができません。その代案として「enum」を用いる方法があります。以下にサンプルコードを示します。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

public enum MMRESULT : uint

{

MMSYSERR\_NOERROR = 0, /\* no error \*/

MMSYSERR\_BASE = 0,

WAVERR\_BASE = 32,

MIDIERR\_BASE = 64,

MMSYSERR\_ERROR = (MMSYSERR\_BASE + 1), /\* unspecified error \*/

MMSYSERR\_BADDEVICEID = (MMSYSERR\_BASE + 2), /\* device ID out of range \*/

// 省略

WAVERR\_BADFORMAT = (WAVERR\_BASE + 0), /\* unsupported wave format \*/

WAVERR\_STILLPLAYING = (WAVERR\_BASE + 1), /\* still something playing \*/

// 省略

MIDIERR\_UNPREPARED = (MIDIERR\_BASE + 0), /\* header not prepared \*/

MIDIERR\_STILLPLAYING = (MIDIERR\_BASE + 1), /\* still something playing \*/

// 省略

}

しかしこの定義では、「MMRESULT.MMSYSERR\_NOERROR」のように利用するときに長い名前になってしまいます。

ここでエラーコードをグループごとに定義すると、「MMRESULT.NOERROR」や「WAVERR.STILLPLAYING」のようになり使いやすくなります。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

public enum MMSYSERR : uint

{

NOERROR = 0, /\* no error \*/

BASE = 0,

ERROR = (BASE + 1), /\* unspecified error \*/

BADDEVICEID = (BASE + 2), /\* device ID out of range \*/

// 省略

}

public enum WAVERR : uint

{

BASE = 32,

BADFORMAT = (BASE + 0), /\* unsupported wave format \*/

STILLPLAYING = (BASE + 1), /\* still something playing \*/

// 省略

}

public enum MIDIERR : uint

{

BASE = 64,

UNPREPARED = (BASE + 0), /\* header not prepared \*/

STILLPLAYING = (BASE + 1), /\* still something playing \*/

// 省略

}

しかしこのままではMMSYSERRのグループがWAVERRやMIDIERRから使えません。MMSYSERRの定義をWAVERRやMIDIERRの中に含める方法もありますが、コーディングが冗長になりスマートな方法とは言えません。

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gif.NET新しい環境へ

C/C++の歴史は長く、関数が戻すエラーコードの定義も様々に発展しました。その結果、全体の構造は複雑で、とても美しいとは言えないものになっています。エラーコードは関数グループ毎に定義され、いくつもの種類が存在します。ここで紹介しているMMRESULTもその中の一つです。これはマルチメディア関連のAPIにだけ適応されるローカルな定義です。

.NETの環境では、関数はエラーコードを戻すより、例外を発生させる手法の方が好まれます。これは例外を使った方が、プログラムがすっきりと書けるという点や、たくさんのエラーコードを管理するより、例外クラスの中で詳細を記述した方が柔軟に対応できるからです。いずれにしろ.NETでは、Win32APIやCOMの複雑なエラーコードと決別しようとしているように見えます。

しかしC#を使ってWin32APIをコールしようとすると、エラーコードとの決別の決心も揺らぎます。上の実装例を見ても分かるように、C#の環境にMMRESULTを定義せざるを得ません。ここでエラーコードに対して、次の2つの考え方があると思われます。

* がんばって移植する
* 移植は最小限に抑える

http://www.cactussoft.co.jp/Sarbo/res/imgList02.gifがんばって移植する

これは、mmsystem.hで記述されたエラーコードをC#環境で使えるよう、コーディングを行うというものです。mmsystem.hは、4000行あまりのファイルなので、がんばれば移植も可能でしょう。

http://www.cactussoft.co.jp/Sarbo/res/imgList02.gif移植は最小限に抑える

仮に移植が可能だとしても、.NETに過去の「しがらみ」を持ち込まないという立場もあるでしょう。この場合、MMRESULTのような変数型は定義しない、という方向になります。

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gifクラスによる定義

ここでエラーコードをクラスを使って定義する方法を紹介します。このエラーコードは、intで定義され、特定の変数型を持ちません。そしてエラーコード群は、「MmSysErr」などのクラスによってラップされます。利用するときは、「MmSysErr.NoError」のような表記となります。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

public class MMSysErr

{

internal const int Base = 0;

public const int NoError = 0; // no error

public const int Error = (Base + 1); // unspecified error

public const int BadDeviceID = (Base + 2); // device ID out of range

// 省略

}

public class WavErr

{

internal const int Base = 32;

public const int BadFormat = (Base + 0); // unsupported wave format

public const int StillPlaying = (Base + 1); // still something playing

// 省略

}

public class MidiErr

{

internal const int Base = 64;

public const int Unprepared = (Base + 0); // header not prepared

public const int StillPlaying = (Base + 1); // still something playing

// 省略

}

ウェーブ関連、MIDI関連のAPIは、MMRESULTを戻すのではなく、intを戻すと定義します。例えば「midiOutLongMsg」関数は、以下のように宣言します。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

[DllImport("winmm.dll")]

public static extern int midiOutLongMsg(IntPtr hmo, ref MidiHdr pmh, int cbmh);

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gif表記の違い

http://www.cactussoft.co.jp/Sarbo/res/imgList02.gifPascal 形式

Win32APIでは、構造体の名前や変数の型は、すべて大文字で表現します。しかし.NETではすべて大文字から成る表記法は用いられません。その代わり単語の頭だけを大文字にし、その他の文字を小文字で表現する「Pascal 形式」が用いられます。一例を示します。

|  |  |
| --- | --- |
| **Win32** | **.NET** |
| MMSYSERR | MMSysErr |
| BADDEVICEID | BadDeviceID |
| STILLPLAYING | StillPlaying |

ここでは.NETに移植するにあたって、ルールに従って変数名を書き換えています。

|  |  |  |
| --- | --- | --- |
| http://www.cactussoft.co.jp/Sarbo/res/imgMemo.gif | |  | | --- | | ただしIO(InputOutput)のように1文字が単語の省略形として意味を持つときは、結果としてすべて大文字になる場合があります。 | |

http://www.cactussoft.co.jp/Sarbo/res/imgList02.gifunsingnedの制限

Win32APIでは、負にならない変数をunsigned属性で表現する習慣があります。以下にWAVEFORMATEX構造体の例を示します。すべての変数が、WORDあるいはDWORDで定義されています。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

typedef struct tWAVEFORMATEX

{

WORD wFormatTag; /\* format type \*/

WORD nChannels; /\* number of channels (i.e. mono, stereo...) \*/

DWORD nSamplesPerSec; /\* sample rate \*/

DWORD nAvgBytesPerSec; /\* for buffer estimation \*/

WORD nBlockAlign; /\* block size of data \*/

WORD wBitsPerSample; /\* number of bits per sample of mono data \*/

WORD cbSize; /\* the count in bytes of the size of \*/

/\* extra information (after cbSize) \*/

} WAVEFORMATEX, \*PWAVEFORMATEX, NEAR \*NPWAVEFORMATEX, FAR \*LPWAVEFORMATEX;

一方、マネージ環境の例を見てみます。以下の定義は、Managed DirectSoundで定義されているウェーブフォーマット構造体の定義です。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

namespace Microsoft.DirectX.DirectSound

{

public struct WaveFormat

{

public WaveFormat();

public int AverageBytesPerSecond { get; set; }

public short BitsPerSample { get; set; }

public short BlockAlign { get; set; }

public short Channels { get; set; }

public WaveFormatTag FormatTag { get; set; }

public int SamplesPerSecond { get; set; }

public override string ToString();

}

}

これを見るとすべての変数はsigned属性で定義されています。チャンネル数やサンプリング周波数など、shortやintのレンジがあればオーバーフローすることがないので、正しく表現できることが分かります。

.NETでは、特にunsigned属性でなければならない理由がない限り、signed属性を用います。

Win32APIの定義を行うとき、元の定義に忠実にunsigned属性で定義するか、.NETの習慣に従ってsigned属性に書き換えるかは、判断の分かれるところでしょう。

ここで象徴的な例として、sizeof()演算子に関する問題を紹介します。

次の関数は、ウェーブ再生で用いるバッファを準備するwaveOutPrepareHeader()関数です。引数の3番目には、WAVEHDR構造体のサイズをバイト単位で指定するよう指示があります。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

MMRESULT waveOutPrepareHeader(

HWAVEOUT hwo,

LPWAVEHDR pwh,

UINT cbwh)

sizeof()は、オブジェクトのサイズを求める演算子で、unsigned intの値を戻します。そのため以下のコードはすんなりと定義に当てはまり、何の問題も起こしません。

http://www.cactussoft.co.jp/Sarbo/res/imgCpp.gif

MMRESULT result = waveOutPrepareHeader(m\_hWaveOut, pWaveHdr, sizeof(WAVEHDR));

一方、.NETでアンマネージ互換の構造体のサイズを求めるときは、Marshal.SizeOf()を用います。WaveHdr構造体のサイズを求めるときは、以下のようになります。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

Marshal.SizeOf(typeof(WaveHdr))

このSizeOf()は、.NETの習慣に従ってintを戻します。

もし、waveOutPrepareHeader関数の3番目の変数のように、サイズを指定する変数をunsigned属性で定義した場合、.NET環境では必ずMarshal.SizeOf()演算子の前に(uint)のキャストが必要になります。

常にキャストが必要になるなら、始めからintで定義しておけば、ということになります。

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gifenumとclass

.NET環境でアンマネージ環境のエラーコードを利用する場合、その移植作業は避けて通れません。MMRESULTはまだ数が少ないので、がんばれば完全な定義を書けるかも知れません。しかし、COMで用いられるHRESULTは規模が非常に大きく、移植作業は困難です。そこで多くの場合、当面使う分だけを記述し、必要があれば追加するという方法を取ります。

C#でenumを用いてMMResultやHResultなどの型を定義すると、元の定義ファイルを修正することなしに新しいエラーコードを追加することはできません。定義ファイルの変更が簡単なら問題ありませんが、複数のスタッフやプロジェクトで共有する場合、定義ファイルのバージョン管理が必要になります。

一方、関数の戻り値をintで定義し、エラーコードをクラスでラップすれば、基本の定義ファイルを修正することなしに拡張することができます。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

public partial class MidiErr

{

internal const int Base = 64;

public const int Unprepared = (Base + 0); // header not prepared

public const int StillPlaying = (Base + 1); // still something playing

// 省略

public const int LastError = (Base + 7); // last error in range

}

元のクラスを拡張するため新しいコードを書きます。ここで鍵になるのが「partial class」です。

http://www.cactussoft.co.jp/Sarbo/res/imgCs.gif

public partial class MidiErr

{

public const int HogeError = (LastError + 1); // My Hoge Error

}

http://www.cactussoft.co.jp/Sarbo/res/imgList03.gifもう一度MMRESULTについて

.NET環境でマルチメディア関連のWin32APIを用いるとき、関数が戻す値は、intとする。というのが.NET流の解釈でしょう。もちろんこれは一つの考え方であり、こうしなければ動かないというものではありません。

.NET環境の中にWin32APIの環境を再現しようとする考えと、.NETの流儀に溶け込むよう解釈し直そうとする2つの立場があります。本技術文書は、後者の立場を取っていますが、どちらを選ぶかは読者の判断です。

[ドキュメントの先頭へ](http://www.cactussoft.co.jp/Sarbo/divManageUse0.html)

[カクタスソフトウェア](http://www.cactussoft.co.jp/) [技術協力](http://www.cactussoft.co.jp/PageTech.html) [資料室](http://www.cactussoft.co.jp/Sarbo/chpAbout.html) [資料室の広場](http://cactussoft.1616bbs.com/bbs/) [SourceForge.jp](http://sourceforge.jp/projects/sarbo/) [お問い合わせ](http://www.cactussoft.co.jp/wp/usersupport/)

## midiInClose 関数

midiInClose

指定された MIDI 入力デバイスをクローズします。

MMRESULT midiInClose(

HMIDIIN hMidiIn

);

パラメータ

*hMidiIn*

MIDI 入力デバイスのハンドルを指定します。関数が成功すると、このハンドルは無効になります。

戻り値

関数が成功すると、MMSYSERR\_NOERROR が返ります。関数が失敗すると、エラーが返ります。返されるエラー値は次のとおりです。

|  |  |
| --- | --- |
| **値** | **説明** |
| MIDIERR\_STILLPLAYING | バッファはまだキューにあります。 |
| MMSYSERR\_INVALHANDLE | 指定されたデバイスハンドルは無効です。 |
| MMSYSERR\_NOMEM | システムはメモリを割り当てられないか、またはロックできません。 |

解説

**midiInAddBuffer**関数を使って送信された入力バッファがあり、それがアプリケーションに返されていない場合、クローズ処理は失敗します。コールバック関数を介して未処理のすべてのバッファを返すには、**midiInReset**関数を使います。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用

参照

[midiInAddBuffer](https://msdn.microsoft.com/ja-jp/library/cc410509.aspx)、[midiInReset](https://msdn.microsoft.com/ja-jp/library/cc410531.aspx)

[印刷](https://msdn.microsoft.com/ja-jp/library/cc410512(d=printer).aspx)[共有](https://msdn.microsoft.com/ja-jp/library/cc410512.aspx)

**このページは役に**

## midiInGetNumDevs 関数

midiInGetNumDevs

システムに存在する MIDI 入力デバイスの数を取得します。

UINT midiInGetNumDevs(VOID);

パラメータ

パラメータはありません。

戻り値

関数が成功すると、システムに存在する MIDI 入力デバイスの数が返ります。戻り値が 0 の場合は、デバイスが存在しないことを示します(エラーがないという意味ではありません）。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用

## midiInGetDevCaps 関数

midiInGetDevCaps

<https://msdn.microsoft.com/ja-jp/library/ms711596.aspx>

指定された MIDI 入力デバイスの性能を調べます。

MMRESULT midiInGetDevCaps(

UINT uDeviceID,

LPMIDIINCAPS lpMidiInCaps,

UINT cbMidiInCaps

);

パラメータ

*uDeviceID*

MIDI 入力デバイスの識別子を指定します。デバイス識別子の値は 0 から存在するデバイス数未満（存在するデバイス数から 1 を引いた数）までの範囲です。また、このパラメータには、適切にキャストされたデバイスハンドルを指定してもかまいません。

*lpMidiInCaps*

デバイスの性能に関する情報が入る **MIDIINCAPS**構造体のアドレスを指定します。

*cbMidiInCaps*

**MIDIINCAPS**構造体のサイズをバイト単位で指定します。*cbMidiInCaps*バイト以下の情報だけが、*lpMidiInCaps*パラメータで指定した場所にコピーされます。*cbMidiInCaps*パラメータが 0 の場合は何もコピーされず、関数は MMSYSERR\_NOERROR を返します。

戻り値

関数が成功すると、MMSYSERR\_NOERROR が返ります。関数が失敗すると、エラーが返ります。返されるエラー値は次のとおりです。

|  |  |
| --- | --- |
| **値** | **説明** |
| MMSYSERR\_BADDEVICEID | 指定されたデバイス識別子は範囲外です。 |
| MMSYSERR\_INVALPARAM | 指定されたポインタまたは構造体は無効です。 |
| MMSYSERR\_NODRIVER | ドライバがインストールされていません。 |
| MMSYSERR\_NOMEM | システムはメモリを割り当てられないか、またはロックできません。 |

解説

システムに存在する MIDI 入力デバイスの数を調べるには、**midiInGetNumDevs**関数を使います。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用  
**Unicode：**Windows NT/2000 は Unicode 版と ANSI 版を実装

参照

**、**[midiInGetNumDevs](https://msdn.microsoft.com/ja-jp/library/cc410520.aspx)

### <http://cpansearch.perl.org/src/NI-S/Audio-1.029/Play/config/mmsystem.h>

### MIDIINCAPS structure (Windows)

MIDIINCAPS structure

The **MIDIINCAPS** structure describes the capabilities of a MIDI input device.

Syntax

C++

typedef struct {

WORD      wMid;

WORD      wPid;

MMVERSION vDriverVersion;

TCHAR     szPname[MAXPNAMELEN];

DWORD     dwSupport;

} MIDIINCAPS;

Members

**wMid**

Manufacturer identifier of the device driver for the MIDI input device. Manufacturer identifiers are defined in [Manufacturer and Product Identifiers](https://msdn.microsoft.com/ja-jp/library/windows/desktop/dd757146(v=vs.85).aspx).

**wPid**

Product identifier of the MIDI input device. Product identifiers are defined in [Manufacturer and Product Identifiers](https://msdn.microsoft.com/ja-jp/library/windows/desktop/dd757146(v=vs.85).aspx).

**vDriverVersion**

Version number of the device driver for the MIDI input device. The high-order byte is the major version number, and the low-order byte is the minor version number.

**szPname**

Product name in a null-terminated string.

**dwSupport**

Reserved; must be zero.

Requirements

|  |  |
| --- | --- |
| **Minimum supported client** | Windows 2000 Professional [desktop apps only] |
| **Minimum supported server** | Windows 2000 Server [desktop apps only] |
| **Header** | Mmsystem.h (include Windows.h) |

See also

[Musical Instrument Digital Interface (MIDI)](https://msdn.microsoft.com/ja-jp/library/windows/desktop/dd743619(v=vs.85).aspx)

[MIDI Structures](https://msdn.microsoft.com/ja-jp/library/windows/desktop/dd757281(v=vs.85).aspx)

#### プラットフォーム呼び出しチュートリアル (C#)

## midiInStart 関数

midiInStart

<https://msdn.microsoft.com/ja-jp/library/ms711592.aspx>

指定された MIDI 入力デバイスで MIDI 入力を開始します。

MMRESULT midiInStart(

HMIDIIN hMidiIn

);

パラメータ

*hMidiIn*

MIDI 入力デバイスのハンドルを指定します。

戻り値

関数が成功すると、MMSYSERR\_NOERROR が返ります。関数が失敗すると、エラーが返ります。返されるエラー値は次のとおりです。

|  |  |
| --- | --- |
| **値** | **説明** |
| MMSYSERR\_INVALHANDLE | 指定されたデバイスハンドルは無効です。 |

解説

この関数はタイムスタンプを 0 にリセットします。リセット後に受信されるメッセージのタイムスタンプの値は、この関数が呼び出された時間に対して相対的な値です。  
システム排他メッセージ以外のすべてのメッセージは、受信されるとクライアントに直接送信されます。システム排他メッセージは、**midiInAddBuffer**関数で提供されたバッファに置かれます。キューにバッファがない場合、システム排他データはクライアントに通知されずに破棄され、入力が継続されます。バッファがいっぱいになった場合、システム排他メッセージが完全に受信された場合、または **midiInReset**関数が使われた場合に、バッファはクライアントに返されます。**MIDIHDR**構造体の **dwBytesRecorded**メンバには、受信されたデータの実際の長さが入ります。  
入力がすでに開始されている場合はこの関数を呼び出しても何も起こらず、関数は 0 を返します。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用

参照

[midiInAddBuffer](https://msdn.microsoft.com/ja-jp/library/cc410509.aspx)、[midiInReset](https://msdn.microsoft.com/ja-jp/library/cc410531.aspx)**、**

## midiInStop 関数

midiInStop

<https://msdn.microsoft.com/ja-jp/library/ms711592.aspx>

指定された MIDI 入力デバイスで MIDI 入力を停止します。

MMRESULT midiInStop(

HMIDIIN hMidiIn

);

パラメータ

*hMidiIn*

MIDI 入力デバイスのハンドルを指定します。

戻り値

関数が成功すると、MMSYSERR\_NOERROR が返ります。関数が失敗すると、エラーが返ります。返されるエラー値は次のとおりです。

|  |  |
| --- | --- |
| **値** | **説明** |
| MMSYSERR\_INVALHANDLE | 指定されたデバイスハンドルは無効です。 |

解説

システム排他メッセージバッファまたはストリームバッファがキューに存在する場合、現在のバッファは処理済みとされます（**MIDIHDR**構造体の **dwBytesRecorded**メンバにはデータの実際の長さが入ります）。ただし、キューにある空のバッファは残され、処理済みとはされません。  
入力が開始される前にこの関数を呼び出しても何も起こらず、関数は 0 を返します。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用

参照

## midiInGetErrorText 関数

midiInGetErrorText

指定されたエラーコードで識別される、エラーの原文記述を取得します。

MMRESULT midiInGetErrorText(

MMRESULT wError,

LPSTR lpText,

UINT cchText

);

パラメータ

*wError*

エラーコードを指定します。

*lpText*

エラーの原文記述が入るバッファのアドレスを指定します。

*cchText*

*lpText*パラメータで指定したバッファの長さを文字数で指定します。

戻り値

関数が成功すると、MMSYSERR\_NOERROR が返ります。関数が失敗すると、エラーが返ります。返されるエラー値は次のとおりです。

|  |  |
| --- | --- |
| **値** | **説明** |
| MMSYSERR\_BADERRNUM | 指定されたエラー番号は範囲外です。 |
| MMSYSERR\_INVALPARAM | 指定されたポインタまたは構造体は無効です。 |
| MMSYSERR\_NOMEM | システムはメモリを割り当てられないか、またはロックできません。 |

解説

エラーの原文記述が指定したバッファよりも長い場合、その記述は切り捨てられます。返されたエラー文字列は常に NULL で終わります。*cchText*パラメータが 0 の場合は何もコピーされず、関数は 0 を返します。すべてのエラー記述の長さは、MAXERRORLENGTH 文字数未満です。

対応情報

**Windows NT/2000：**Windows NT 3.1 以降  
**Windows 95/98：**Windows 95 以降  
**ヘッダー：**mmsystem.h 内で宣言  
**インポートライブラリ：**winmm.lib を使用  
**Unicode：**Windows NT/2000 は Unicode 版と ANSI 版を実装

# ＠IT：.NET TIPS Win32 APIやDLL関数を呼び出すには？ - C#

**NET TIPS：Win32 APIやDLL関数を呼び出すには？**

求める機能が.NET Frameworkのクラス・ライブラリに存在しない場合、その代わりに、.NETプログラムからWindowsシステムのAPIであるWin32 APIを直接呼び出せる。C#およびVB.NETでの実現方法を解説する。

[泉祐介，デジタルアドバンテージ]

[「.NET TIPS」のインデックス](http://www.atmarkit.co.jp/ait/subtop/features/dotnet/dotnettips_index.html)

[連載目次](http://www.atmarkit.co.jp/ait/subtop/features/dotnet/dotnettips_index.html)

　.NET Frameworkのクラス・ライブラリには、アプリケーション構築用として非常に多くの機能が用意されている。しかし、WindowsシステムのAPIであるWin32 APIの機能のすべてが、クラス・ライブラリとして用意されているわけではない。このような機能を.NETプログラムから使用する必要がある場合には、ここで紹介する方法によりWin32 APIを直接呼び出すことができる。また、.NET以前の環境でDLLファイルとして作成されたライブラリも同様に呼び出し可能である。

　.NET Framework上で動作するプログラムから、従来形式の（.NETアセンブリなどではない）DLLファイルがエクスポートしている関数を呼び出すには、DllImport属性（System.Runtime.InteropServices名前空間）を利用し、その関数が外部にあることをあらかじめ宣言しておけばよい。これは、Win32 APIを呼び出す場合にも同様である。

　例えば、マシンに内蔵されているブザー（高機能なサウンド・デバイスではない）からビープ音を鳴らすBeepというWin32 APIがWindowsには用意されている。このAPIの書式は次のとおりである（パラメータなどの詳細については[MSDNドキュメント](http://www.microsoft.com/japan/msdn/library/?url=/japan/msdn/library/ja/jpdebug/html/_win32_beep.asp?frame=true)を参照）。

BOOL Beep(DWORD dwFreq, DWORD dwDuration);

　このAPIをC#で利用するには、適当なクラスの内部に以下のような宣言を加えればよい。

[DllImport("kernel32.dll")]  
extern static bool Beep(uint dwFreq, uint dwDuration);

　この宣言により、通常のメソッドと同様に、プログラム内からBeep関数を呼び出すことができるようになる。

　DllImport属性は、プログラムから利用する関数をエクスポートしているDLLのファイル名を指定するための属性である。Win32 APIに対するDLLファイル名は、MSDNドキュメントにある各APIの解説で、「インポート ライブラリ」という項目に記述されたファイル名から知ることができる。具体的には、kernel32.libと記述されていればkernel32.dll、user32.libであればuser32.dll、gdi32.libであればgdi32.dllとなる。一般に、ほとんどの場合はインポート・ライブラリの欄に記述されたファイル名の拡張子をdllに変更したものを指定すればよい。先のBeep関数は、ドキュメントによればインポート・ライブラリはkernel32.libであるため、DllImport属性のパラメータとしてkernel32.dllを指定している。

　また、DllImport属性を付けたWin32 APIやDLL関数の宣言では、関数の実体が外部にあることを表すextern修飾子と、静的なメンバであることを表すstatic修飾子を必ず指定する。なお、ここではpublicやprivateなどのアクセス修飾子は省略したが、必要であれば指定することもできる。

　BOOLやDWORDといったパラメータや戻り値の型に関しては、対応するC#の型を指定する必要がある。例えば、前出の例ではBOOLに対してbool、DWORDに対してuintを指定している。主要なWin32 APIでの型名と、それに対応するC#の型は以下の表のようになっている。もちろん、この対応表はWin32 APIではない一般的なDLL関数に対しても利用できる。

| **APIでの型名 （括弧内は対応するC言語の型）** | **対応するC#の型 （括弧内は.NET Frameworkでの型名）** |
| --- | --- |
| **HANDLE (void \*)** | System.IntPtr |
| **BYTE (unsigned char)** | byte (System.Byte) |
| **SHORT (short)** | short (System.Int16) |
| **WORD (unsigned short)** | ushort (System.UInt16) |
| **INT (int) LONG (long)** | int (System.Int32) |
| **UINT (unsigned int)  DWORD, ULONG (unsigned long)** | uint (System.UInt32) |
| **BOOL (long)** | bool (System.Boolean) |
| **CHAR (char)** | char (System.Char) |
| **LPSTR (char \*) LPWSTR (wchar\_t \*)** | System.Text.StringBuilder |
| **LPCSTR (const char \*) LPCWSTR (const wchar\_t \*)** | string (System.String) |
| **FLOAT (float)** | float (System.Single) |
| **DOUBLE (double)** | double (System.Double) |
|  | |

**Win32 APIでの型名と対応するC#の型**  
WindowsのDLL（Win32 API）と.NET Frameworkとでは型の管理方法が違うため、実際には型の相互変換（マーシャリング）が行われる。なお、BOOL型の実体はLONG型と同じなので、boolの代わりにintを指定することも可能である。

　Win32 APIの具体的な利用例として、Beep関数を利用したサンプル・プログラムを1つ挙げておこう。以下のリストは、マシン内蔵のスピーカから「ドレミファソラシド」を鳴らすプログラムである。マシン内蔵のブザーを使用するため、音を鳴らすといってもサウンド・デバイスは必要ない。

// beep.cs  
  
using System;  
using System.Runtime.InteropServices;  
  
class BeepProgram {  
  [DllImport("kernel32.dll")]  
  private extern static bool Beep(uint dwFreq, uint dwDuration);  
  
  private static void Main() {  
    Beep(262, 500);  // ド  
    Beep(294, 500);  // レ  
    Beep(330, 500);  // ミ  
    Beep(349, 500);  // ファ  
    Beep(392, 500);  // ソ  
    Beep(440, 500);  // ラ  
    Beep(494, 500);  // シ  
    Beep(523, 500);  // ド  
  }  
}  
  
// コンパイル方法: csc beep.cs

**マシン内蔵のスピーカから「ドレミファソラシド」を鳴らすプログラム（beep.cs）**  
[beep.csのダウンロード](http://www.atmarkit.co.jp/fdotnet/dotnettips/024w32api/beep.cs)  
[beep.vb（Visual Basic .NET版）のダウンロード](http://www.atmarkit.co.jp/fdotnet/dotnettips/024w32api/beep.vb)

**参照渡しのパラメータを含むDLL関数の場合**

　先のBeep関数では、パラメータとして2つの数値を渡すだけの単純なDLL呼び出しで済むが、一部または全部のパラメータが参照渡しになっているAPIやDLL関数も存在する。

　例えば、マウスボタンの数を取得するWin32 APIであるGetNumberOfConsoleMouseButtons関数の書式は次のようになっている。

BOOL GetNumberOfConsoleMouseButtons(LPDWORD lpNumberOfMouseButtons);

　パラメータのlpNumberOfMouseButtonsはLPDWORD型（DWORD型の値へのポインタ）であるが、実際にはDWORD型の値を参照渡しで渡すことを意味している。

　このような参照渡しのパラメータを含む関数を宣言するには、参照渡しになっているパラメータをrefパラメータとして宣言すればよい。よって、GetNumberOfConsoleMouseButtons関数の宣言は以下のようになる。

[DllImport("kernel32.dll")]  
extern static bool GetNumberOfConsoleMouseButtons(ref uint lpNumberOfMouseButtons);

　ただし、呼び出し側から値を渡す必要がないことが分かっている場合はoutパラメータと宣言しても構わない。先のGetNumberOfConsoleMouseButtons関数も値を渡す必要はないので、以下のように宣言することもできる。

[DllImport("kernel32.dll")]  
extern static bool GetNumberOfConsoleMouseButtons(out uint lpNumberOfMouseButtons);

**定数を使用するDLL関数の場合**

　Win32 APIやDLL関数を利用する際に、関数のパラメータとして何らかの定数を指定する場合がある。特にWin32 APIに関しては、それらの定数がプラットフォームSDKに含まれるヘッダ・ファイル（windows.hなど）で定義されているが、C#ではこれらのヘッダ・ファイルを扱うことができないため、必要に応じて定数を自分で定義しなければならない。

　Win32 APIやDLL関数に定数を渡す方法はいくつか考えられるが、必要な定数をconstキーワードで定義する方法が最も一般的だろう。例えば、以下のサンプル・プログラムは、IsProcessorFeaturePresentというWin32 APIを利用して、プロセッサが[MMX命令](http://www.atmarkit.co.jp/icd/root/85/5784585.html)セットをサポートしているかどうかを調べるプログラムである。IsProcessorFeaturePresent関数はプロセッサが特定のプロセッサ機能をサポートしているかどうかを調べるAPIであり、プロセッサ機能の種類を表す定数をパラメータにとる。このプログラムでは、SSE命令セットを表すPF\_XMMI\_INSTRUCTIONS\_AVAILABLE定数を宣言し、それをパラメータとして渡している（詳細は[MSDNドキュメント（英語）](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/isprocessorfeaturepresent.asp)を参照）。なお、定数の実際の値（この場合は6）を知るには、プラットフォームSDKに含まれるヘッダ・ファイルを参照する必要がある。

// ssechk1.cs  
  
using System;  
using System.Runtime.InteropServices;  
  
class SSECheck1 {  
  [DllImport("kernel32.dll")]  
  private extern static bool IsProcessorFeaturePresent  
    (uint ProcessorFeature);  
  
  // 定数の宣言  
  private const uint PF\_XMMI\_INSTRUCTIONS\_AVAILABLE = 6;  
  
  private static void Main() {  
    if(IsProcessorFeaturePresent(PF\_XMMI\_INSTRUCTIONS\_AVAILABLE)) {  
      Console.WriteLine("SSE is available.");  
    } else {  
      Console.WriteLine("SSE is not available.");  
    }  
  }  
}  
  
// コンパイル方法: csc ssechk1.cs

**プロセッサのSSE命令セットのサポートの有無を調べるプログラム（ssechk1.cs）**  
[ssechk1.csのダウンロード](http://www.atmarkit.co.jp/fdotnet/dotnettips/024w32api/ssechk1.cs)  
[ssechk1.vb（Visual Basic .NET版）のダウンロード](http://www.atmarkit.co.jp/fdotnet/dotnettips/024w32api/ssechk1.vb)

　独自の列挙型を宣言し、その列挙型を関数のパラメータや戻り値として宣言する方法もある。例として、IsProcessorFeaturePresent関数のパラメータを列挙型として宣言したサンプル・プログラムを以下に示す。

// ssechk2.cs  
  
using System;  
using System.Runtime.InteropServices;  
  
// 定数の宣言  
enum ProcessorFeatures : uint {  
  PF\_FLOATING\_POINT\_PRECISION\_ERRATA = 0,  
  PF\_FLOATING\_POINT\_EMULATED = 1,  
  PF\_COMPARE\_EXCHANGE\_DOUBLE = 2,  
  PF\_MMX\_INSTRUCTIONS\_AVAILABLE = 3,  
  PF\_PPC\_MOVEMEM\_64BIT\_OK = 4,  
  PF\_ALPHA\_BYTE\_INSTRUCTIONS = 5,  
  PF\_XMMI\_INSTRUCTIONS\_AVAILABLE = 6,  
  PF\_3DNOW\_INSTRUCTIONS\_AVAILABLE = 7,  
  PF\_RDTSC\_INSTRUCTION\_AVAILABLE = 8,  
  PF\_PAE\_ENABLED = 9,  
  PF\_XMMI64\_INSTRUCTIONS\_AVAILABLE = 10  
}  
  
class SSECheck2 {  
  [DllImport("kernel32.dll")]  
  private extern static bool   
    IsProcessorFeaturePresent(ProcessorFeatures ProcessorFeature);  
  
  private static void Main() {  
    if(IsProcessorFeaturePresent  
        (ProcessorFeatures.PF\_XMMI\_INSTRUCTIONS\_AVAILABLE)) {  
      Console.WriteLine("SSE is available.");  
    } else {  
      Console.WriteLine("SSE is not available.");  
    }  
  }  
}  
  
// コンパイル方法: csc ssechk2.cs

**プロセッサのSSE命令セットのサポートの有無を調べるプログラム（ssechk2.cs）**  
[ssechk2ssechk2のダウンロード](http://www.atmarkit.co.jp/fdotnet/dotnettips/024w32api/ssechk2.cs)  
[ssechk2.vb（Visual Basic .NET版）のダウンロード](http://www.atmarkit.co.jp/fdotnet/dotnettips/024w32api/ssechk2.vb)

　この程度のプログラムでは、列挙型を定義するメリットはあまりなく、むしろ列挙体を定義することがかえって手間になっている。しかし、特に複数の種類の定数を利用する場合は、パラメータを列挙型として宣言しておけば、その列挙型のメンバ以外の値を直接パラメータに渡すことができなくなるので、コーディング上のミスを抑えることが可能になる。また、Visual Studio .NETのIntelliSence機能では、パラメータの型としてこの列挙型が表示されるため、どの定数を指定すればよいかが一目で分かるようになる。これは戻り値についても同様である。必要に応じてconstによる方法と使い分けるとよいだろう。

　なお、文字列や構造体をパラメータに指定するDLL関数呼び出しについては、「[TIPS：Win32 APIやDLL関数に文字列や文字列バッファを渡すには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/025w32string/w32string.html) 」や「[TIPS：Win32 APIやDLL関数に構造体を渡すには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/026w32struct/w32struct.html) 」を参照していただきたい。

**カテゴリ：**クラス・ライブラリ　**処理対象：**Win32 API  
**使用ライブラリ：**DllImport属性（System.Runtime.InteropServices名前空間）  
**関連TIPS：**[Win32 APIやDLL関数に文字列や文字列バッファを渡すには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/025w32string/w32string.html)  
**関連TIPS：**[Win32 APIやDLL関数に構造体を渡すには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/026w32struct/w32struct.html)

**■この記事と関連性の高い別の.NET TIPS**

* [実行ファイルからアプリケーションのアイコンを取得するには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/280getappicon/getappicon.html)
* [Win32 APIやDLL関数に文字列や文字列バッファを渡すには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/025w32string/w32string.html)
* [Win32 APIやDLL関数に構造体を渡すには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/026w32struct/w32struct.html)
* [ドライブをフォーマットするためのダイアログを表示するには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/332formatdrive/formatdrive.html)
* [Win32 API呼び出しを手軽に記述するには？](http://www.atmarkit.co.jp/fdotnet/dotnettips/166pinvoke/pinvoke.html)

[「.NET TIPS」のインデックス](http://www.atmarkit.co.jp/ait/subtop/features/dotnet/dotnettips_index.html)

「[.NET TIPS](http://www.atmarkit.co.jp/ait/subtop/features/dotnet/dotnettips_index.html)」

# 意外と知られていない Windows のエラーコードの数々 - 岩田 雅樹 のブログ - Site Home - MSDN Blogs

意外と知られていない Windows のエラーコードの数々

[MSDN検索術](http://msdn.microsoft.com/ja-jp/kensaku.aspx)が公開されたのでそれに合わせてメモなど。  
開発者の皆様は (私もそうなのですが･･･) 毎日のようにとんでもない量のエラーメッセージを目にされていると思います。

ところで、これらのエラーメッセージ、エラーコードがどのような区分になっているかご存知でしょうか？今回は、よく目にするものと、その見分け方について簡単にご紹介させていただきたいと思います。

エラーを正しく理解することは正確な報告を可能にし、また問題解決にも役立ちます。

1. .NET Framework における例外

こちらは (見た目は) わかりやすいですよね。例えば何か存在しないファイルを開こうとすると…

Unhandled Exception: System.IO.FileNotFoundException: Could not find file 'C:\Demos\ExceptionDemo\bin\Debug\何か。'.  
File name: 'C:\Demos\ExceptionDemo\bin\Debug\何か。'  
   at System.IO.\_\_Error.WinIOError(Int32 errorCode, String maybeFullPath)  
…  
   at System.IO.StreamReader..ctor(String path, Boolean detectEncodingFromByteOrderMarks)  
   at ExceptionDemo.Program.Main(String[] args) in C:\Demos\ExceptionDemo\Program.cs:line 12

このような例外が発生すると思います。全ての例外は System.Exception クラスを継承しなければならないため、System.Exception クラスを通して、

Data …追加情報。この例では空のコレクション  
HelpLink …ヘルプへの URN。この例では null  
InnerException …この例外を発生させる元となった例外。この例では null  
Message …例外の説明。この例では “Could not find file 'C:\Demos\ExceptionDemo\bin\Debug\何か。'.”  
Source …例外を投げたアプリケーションやオブジェクト。この例では “mscorlib” つまり、.NET Framework の基本ライブラリ  
StackTrace …例外が投げられるまでの呼び出しの流れ。この例では “at System.IO… ~ Program.cs:line 12”  
TargetSite …例外を投げたメソッド。この例では {Void WinIOError(Int32, System.String)}

さらに、今回は System.Exception –> … –> System.IO.FileNotFoundException と継承されているので、FileNotFoundException 内の

FileName …ファイル名。この場合は @”C:\Demos\ExceptionDemo\bin\Debug\何か。”

などが利用できます。さて、問題解決には何が重要になるでしょうか？「例外が発生した！」と報告しても誰にも原因はわからないですよね。  
例外の場合、例外の種類、StackTrace と引数が重要になります。どの関数を呼んだ事で例外が発生したのかを知り、呼び出した関数への引数に問題がなかったかがわかれば再現させることが容易になります。例えばこの場合ですと、

「Program.cs の 12行目、System.IO.StreamReader..ctor(String path, Boolean detectEncodingFromByteOrderMarks) の呼び出しにおいて System.IO.FileNotFound が発生、引数には “何か。” が与えられていた。」と報告すれば、「”何か。” という存在しないファイルを読み出そうとしたのだな」ということが確実に相手に伝わります。

複雑な問題においては、InnerException にも着目しましょう。根本的な原因がそこに書かれている場合もあります。フォーラムなどで質問する際には例外に含まれる情報を活用いただければと思います。

2. Win32, COM におけるエラーコード

.NET における例外はこのくらいにして Win32, COM (Component Object Model) においてはどのようなエラーが生じるのでしょうか。Native のプログラムを書いていると 0xC0000005 というエラーコードに遭遇することが多いと思いますが、これは Win32 における一般保護例外、つまりアクセスできない領域に対するエラーのコードです。はて、なぜこんなに長いエラーコードになっているのでしょう？実はエラーコードの先頭には意味がついています。エラーコードの 32bit の中身を分解すると以下のようになります。

Win32 の場合

| Severity Code (2bit) | Customer code flag (1bit) | Reserved (1bit) | Facility Code (12bit) | Code (16bit) |

COM の場合

| Severity Code (1bit) | Reserved (4bit) | Facility Code (11bit) | Code (16bit) |

ここで、Severity Code は重要度を表しています。Win32 の場合、00b が成功、01b が情報を含むもの、10b が警告、11b がエラーとなっています。Windows が返すエラーには Customer Code Flag は設定されていないので先頭 4bit …つまり、16進数表記をした際の最初の一文字がエラーコードの重要度となります。0xC0000005 は ‘C’ で始まっているので 0xC = 1100b つまりエラーを示しています。ただ、警告や情報は滅多に見かけないので 0xC0123456 といったエラーコードを目にすることになります。

一般保護例外 0xC0000005 は、Severity = 11b から「エラー」であり、Code = 5 (ERROR\_ACCESS\_DENIED) から「Access is denied. (アクセスが拒否された)」ことが読み取れるのです。

さて、COM の場合は先頭が少し異なりますよね？ Severity Code が 1bit となっています。なので COM のエラーコード (HRESULT) は 0x80123456 といった形になります。

ここがポイントです！つまり、たいていの場合、0xC で始まっているものは Win32 のエラーコード、0x8 で始まっているものは COM のエラーコードという見分け方ができるのです。もし 10進数でエラーコードが –1073741824 のような負の大きな値で –1 始まりのものは Win32, –2147483648 のような –2 始まりのものは COM のエラーコードである可能性が高いです。

これは電卓をプログラマモード (関数モード) にすることで変換することが可能です。エラーコードを検索しても出てこない場合には、16進数のコードを 10進数に直してみる、あるいはその逆を行うことで解決策が引っかかる可能性が高くなります。

また、下位 16bit にエラーの内容を示すコードが入っているため、そこだけ抜き出して Error Lookup ツールなどを利用すると何かヒントが得られるかも知れません。

3. 一般的なエラーコード

C言語の関数の結果、またイベントログに記載されるエラーコードは、このような長いエラーコード形式ではないのでせいぜい数千までの値を取ることが多いと思います。そのような場合には 16進数で検索せず、10進数で検索するほうがよい結果が得られる可能性が高いと思います。Windows SDK に含まれる、WinError.h を開いて検索するというのも一つの手です。

まとめ。

例外など、型情報が得られるものについては例外の種類を見分ける。

8 で始まっている、または –21… という大きな負の値なら COM のエラーコードであることを疑う。  
C で始まっている、または –10… という大きな負の値なら Win32 のエラーコードであることを疑う。  
覚えられる範囲の数字なら一般的なエラーコードを疑う。

もしかしたら 10進数で負の値になっているものは 16進数に変換することで解決策が見つかるかも知れません。

このように、ちょっとした見分けかたを覚えておくだけでもエラーコードからより多くの情報、または検索に頼った問題解決のヒントが得られます。

# 訂正しました。mayukiさん、つーささん、ありがとうございます！

Comments (3)

# MIDI Table

|  |  |  |
| --- | --- | --- |
| **MIDI Table** | | |
| [Prev](http://fmslogo.sourceforge.net/manual/command-midimessage.html) | **MIDI Commands** | [Next](http://fmslogo.sourceforge.net/manual/midi-instrument.html) |

**MIDI Table**

This table defines some messages that are legal for [MIDIMESSAGE](http://fmslogo.sourceforge.net/manual/command-midimessage.html).

| **Command Name** | **Status Byte** | **Data Byte 1** | **Data Byte 2** |
| --- | --- | --- | --- |
| Note Off | 128 + Channel | 0-127 Pitch | 0-127 Velocity |
| Note On | 144 + Channel | 0-127 Pitch | 0-127 Velocity |
| Poly Pressure | 160 + Channel | 0-127 Pitch | 0-127 Pressure |
| Control Change | 176 + Channel | 0 Undefined | 0-127 MSB |
| Control Change | 176 + Channel | 1 Modulation Wheel | 0-127 MSB |
| Control Change | 176 + Channel | 2 Breath Controller | 0-127 MSB |
| Control Change | 176 + Channel | 3 After Touch | 0-127 MSB |
| Control Change | 176 + Channel | 4 Foot Controller | 0-127 MSB |
| Control Change | 176 + Channel | 5 Portamento Time | 0-127 MSB |
| Control Change | 176 + Channel | 6 Data Entry | 0-127 MSB |
| Control Change | 176 + Channel | 7 Main Volume | 0-127 MSB |
| Control Change | 176 + Channel | 8-31 Undefined | 0-127 MSB |
| Control Change | 176 + Channel | 32-63 LSB of 0-31 | 0-127 MSB |
| Control Change | 176 + Channel | 64 Damper Pedal | 0:off 127:on |
| Control Change | 176 + Channel | 65 Portamento | 0:off 127:on |
| Control Change | 176 + Channel | 66 Sostenuto | 0:off 127:on |
| Control Change | 176 + Channel | 67 Soft Pedal | 0:off 127:on |
| Control Change | 176 + Channel | 68-92 Undefined | 0:off 127:on |
| Control Change | 176 + Channel | 93 Chorus | 0:off 127:on |
| Control Change | 176 + Channel | 94 Celeste | 0:off 127:on |
| Control Change | 176 + Channel | 95 Phaser | 0:off 127:on |
| Control Change | 176 + Channel | 96 Data Entry + 1 | 0:off 127:on |
| Control Change | 176 + Channel | 97 Data Entry - 1 | 0:off 127:on |
| Control Change | 176 + Channel | 98-121 Undefined | 0:off 127:on |
| Control Change | 176 + Channel | 122 Local Control | 0-127 |
| Control Change | 176 + Channel | 123 All Notes Off | 0 |
| Control Change | 176 + Channel | 124 Omni Mode off | 0-15 |
| Control Change | 176 + Channel | 125 Omni Mode on | 0 |
| Control Change | 176 + Channel | 126 Mono on/Poly off | 0 |
| Control Change | 176 + Channel | 127 Poly on/Mono off | 0 |
| Program Change | 192 + Channel | 0-127 Program | Not used |
| Channel Pressure | 208 + Channel | 0-127 Pressure | Not used |
| Pitch Wheel | 224 + Channel | 0-127 LSB | 0-127 MSB |
| System Exclusive | 240 | 0-127 Id Code | Any number of bytes |
| Undefined | 241 | Not used | Not used |
| Song Position | 242 | 0-127 LSB | 0-127 MSB |
| Song Select | 243 | 0-127 Song | Not used |
| Undefined | 244 | Not used | Not used |
| Undefined | 245 | Not used | Not used |
| Tune Request | 246 | Not used | Not used |
| End of Exclusive | 247 | Not used | Not used |
| Timing Clock | 248 | Not used | Not used |
| Undefined | 249 | Not used | Not used |
| Start | 250 | Not used | Not used |
| Continue | 251 | Not used | Not used |
| Stop | 252 | Not used | Not used |
| Undefined | 253 | Not used | Not used |
| Active Sensing | 254 | Not used | Not used |
| System Reset | 255 | Not used | Not used |

# README.mdファイル。マークダウン記法まとめ | codechord

[README.mdファイル。マークダウン記法まとめ](http://codechord.com/2012/01/readme-markdown/)

Posted 1月 21st, 2012 by codechord. [6 Comments](http://codechord.com/2012/01/readme-markdown/#comments)

つい先日から、ようやくgithubを使い始めました。  
gitのことはあまり勉強してないので右も左もわからず、とりあえず、公開されてるレポジトリのクローンを作りまくってニヤニヤ閲覧しています。

で、そろそろ自分もリポジトリを作りたいなと思って、作り方を調べてたんですが、  
「**README.md**」ファイルというものの存在を知りました。

拡張子「.md」？何の事かさっぱりわからず、ドットモジュールって勝手に思ってたんですけど、全然違いました。「マークダウン」の略でした///

**マークダウンとは**

マークダウンファイルとは何ぞや。調べました。  
普通のテキストファイルを、ちょっとした法則にならって書くだけで、HTMLマークアップせずに、自動的にHTMLコードとして変換して出力してくれるというものでした。  
このちょっとした法則がマークダウン記法っていう。

READMEファイルは、このマークダウン記法で書くと、わかりやすいREADMEが作れるよ。ということですが、  
要は、**READMEを書くときに、読む人にもわかりやすいように構成考えて書こうね。**  
って事だと思います。

いろんなところで使われてるみたいで、ホント今更知った。  
ちなみに、githubでのマークダウン記法はGFM（GitHub Flavored Markdown）と呼ばれてるらしい。

**マークダウン記法**

さて、それではまとめていきます。  
ざっと調べただけなので、不足や間違いがあるかもしれません。

**段落**

空行をあける

1段落目

（空行をはさむ）

2段落目

**改行**

行末に2つのスペースを入れる。

1行目 ←行末に半角スペース2つ

2行目

**強調**

・強調。アスタリスク（\*）またはアンダーバー(\_)で囲う

\*強調\* または \_強調\_

・強い強調。アスタリスク2つ（\*\*）またはアンダーバー2つ(\_\_)で囲う

\*\*強い強調\*\* または \_\_強い強調\_\_

**コード表示（等幅フォントで表示）**

・インラインのコード。バッククォーテーションで囲う

`$hoge = 1`

・複数行のコード。半角スペースを4つ行頭に

    ←半角スペース4つ　$hoge1 = 1 //1行目

    ←半角スペース4つ　$hoge2 = 2 //2行目

    ←半角スペース4つ　$hoge3 = 3 //3行目

**リスト**

行頭にアスタリスク(\*)か、番号ピリオド（1. ）

\* リスト1

\* リストリスト1-2

\* リスト2

1. 順番つきリスト

2. 順番つきリスト

3. 順番つきリスト

**見出し**

行頭の#の個数でH1～H6を表す。

# H1見出し

## H2見出し

#### H4見出し

または、

H1見出し（イコール行を挿入。）

==================

H2見出し（ハイフン行を挿入。）

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

**引用（blockquote）**

メールの引用の感じ。途中で改行しても、ダブルクォーテーション内は改行されない。入れ子もOK

> "引用本文引用本文引用本文引用本文

引用本文引用本文引用本文引用本文引用本文引用本文。"

> 引用本文引用本文引用本文引用本文

> 引用本文引用本文引用本文引用本文引用本文引用本文。

> > 入れ子入れ子

> > 入れ子入れ子

> 引用本文引用本文引用本文引用本文

> \* リスト

> 引用本文引用本文引用本文引用本文

> 1. 番号リスト

> 引用本文引用本文引用本文引用本文

> $hgoe = 1 コードを書いてみたり

> 引用本文引用本文引用本文引用本文

**リンク**

記法は3種。リンクのタイトルは省略可能。

■自動リンク  
アングルブラケット< >でURLを囲う。メアドでも可能

<http://example.com>

<example@example.com>

■通常のインライン記法

[リンクのテキスト](リンクのアドレス "リンクのタイトル")

■外部参照リンク（段落の外でリンクをまとめれる）

[リンクのテキスト][linkref]

[linkref]: リンクのアドレス "リンクのタイトル"

・実際の例：

I get 10 times more traffic from [Google][] than from

[Yahoo][] or [MSN][].

[google]: http://google.com/        "Google"

[yahoo]:  http://search.yahoo.com/  "Yahoo Search"

[msn]:    http://search.msn.com/    "MSN Search"

**画像**

先頭にビックリマーク

![Alt text](/path/to/img.jpg)

![Alt text](/path/to/img.jpg "Optional title")

・リンクと同じように参照表記も可能

![Alt text][id]

[id]: url/to/image  "Optional title attribute"

**水平線**

3つ以上のハイフン、アスタリスク、アンダースコアをならべる。間にスペースをいれても良い

\* \* \*

\*\*\*

\*\*\*\*\*

- - -

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

**その他の注意**

&マークなどは「&amp;」に変換しておく。

■エスケープしておくべき箇所

* 行頭でのピリオドをリストではなく、通常のように表示させたい場合。
* アスタリスクを通常で表示させたい場合。
* バッククォートを表示させたい場合は、連続バッククォート。“あいう`えお“

**エスケープ一覧**

\   backslash

`   backtick

\*   asterisk

\_   underscore

{}  curly braces

[]  square brackets

()  parentheses

#   hash mark

+   plus sign

-   minus sign (hyphen)

.   dot

!   exclamation mark

**まとめ**

結局のところ、実際に、自分で書いてみないとわかりません。  
こちらのページでリアルタイムで反映されるwebエディタがあるので、ちょっと触ってみるだけで、雰囲気がつかめるかと思います。

* <http://pamgau.net/showdown/>

**参考させていただいたサイト**

* <http://tokkono.cute.coocan.jp/blog/slow/index.php/programming/markdown-skills-for-github-beginners/>
* <http://ja.wikipedia.org/wiki/Markdown>
* <http://blog.2310.net/archives/6>

githubに特化したマークダウンは本家を見るべきですね。

* <http://github.github.com/github-flavored-markdown/>

http://www17.a8.net/0.gif?a8mat=1ZIPZD+2RFESY+348+TS3OH http://www13.a8.net/0.gif?a8mat=1ZIPZD+2TT582+D8Y+69HA9

Posted in [SCM](http://codechord.com/category/scm/), [git](http://codechord.com/category/scm/git/) Tagged as [github](http://codechord.com/tag/github/), [まとめ](http://codechord.com/tag/%e3%81%be%e3%81%a8%e3%82%81/), [マークダウン](http://codechord.com/tag/%e3%83%9e%e3%83%bc%e3%82%af%e3%83%80%e3%82%a6%e3%83%b3/)

**6 Responses to README.mdファイル。マークダウン記法まとめ**

1. [はじめての Github で自作のライブラリを公開するためのリポジトリ作成や設定まとめ | ウェブル](http://weble.org/2012/05/09/github-beginner) says: [2012/05/09 10:00 AM](http://codechord.com/2012/01/readme-markdown/#comment-106)

[…] README.mdファイル。マークダウン記法まとめ | codechord […]

1. [9ra10 » MarkDownを勉強し始める](http://9ra10.com/markdown%e3%82%92%e5%8b%89%e5%bc%b7%e3%81%97%e5%a7%8b%e3%82%81%e3%82%8b/) says: [2013/03/11 11:52 AM](http://codechord.com/2012/01/readme-markdown/#comment-604)

[…] <http://codechord.com/2012/01/readme-markdown/> <http://bamka.info/3765/><http://blog.2310.net/archives/6> […]

1. [コーディング規約を作ろう | Webクリエイターボックス](http://www.webcreatorbox.com/webinfo/coding-guideline/) says: [2013/08/09 7:27 AM](http://codechord.com/2012/01/readme-markdown/#comment-1860)

[…] 詳しい説明は「README.mdファイル。マークダウン記法まとめ」がわかりやすかったです。 […]

1. [README.mdファイル。マークダウン記法まとめ | codechord | さとういちろうのへや](http://sato21.sakura.ne.jp/any/archives/2993) says:[2015/05/14 9:08 AM](http://codechord.com/2012/01/readme-markdown/#comment-87459)

[…] 情報源: README.mdファイル。マークダウン記法まとめ | codechord […]

1. [Markdown記法を使ってみる | cly7796.net](http://cly7796.net/wp/css/try-to-use-the-markdown-notation/) says: [2015/07/12 11:48 PM](http://codechord.com/2012/01/readme-markdown/#comment-94103)

[…] README.mdファイル。Markdown記法まとめ | codechord […]

1. [git/githubのお勉強 | よっしーの冒険](http://yosshi.snowdrop.asia/2015/08/29/gitgithub%e3%81%ae%e3%81%8a%e5%8b%89%e5%bc%b7/) says: [2015/08/29 9:43 AM](http://codechord.com/2012/01/readme-markdown/#comment-101614)

[…] README.mdの書き方 <http://codechord.com/2012/01/readme-markdown/> […]

**コメントを残す**

窗体顶端

メールアドレスが公開されることはありません。 \* が付いている欄は必須項目です

# 白執事の徒然なる日々 VB.NET で USBデバイス を検出する！