SharpMIDI

MIDI library for .NET

Manual

Version 1.0

Contents

1. Adding SharpMIDI to your project

1.1. Downloading

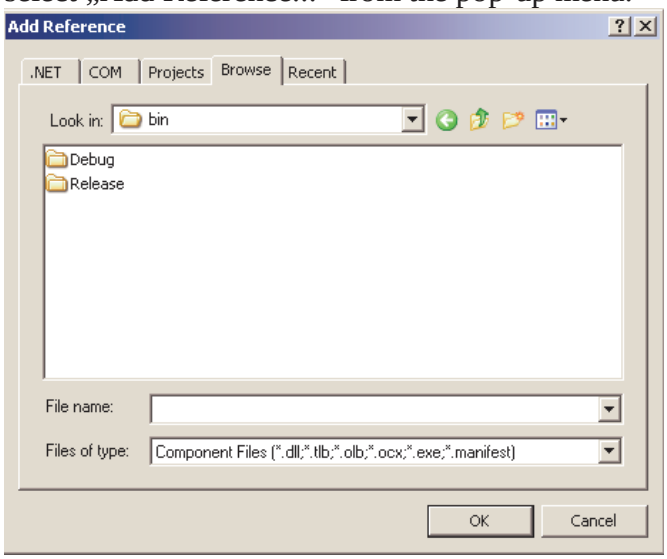
You can download the SharpMIDI binaries freely, from this web page:
http://bolner.hu/sharpmidi
After extracting the compressed file, you will find these three files:

Filename	Bitness	Language	Description
SharpMidi.dll	Flexible (32/64)	C#	You use only this assembly directly. (.NET 3.5)
SharpMidiCPP32.dll	32 bit	C++	32 bit native dll to access Windows API, used by SharpMidi.dll
SharpMidiCPP64.dll	64 bit (x64, AMD64)	C++	64 bit native dll to access Windows API, used by SharpMidi.dll

Copy these files into the same folder as your projects output folder, where the executable is placed. It's strongly advised to put both native dll's aside the SharpMIDI.dll, even if you aimed to develop only for a 32, or only for a 64 bit system, to achive a higher flexibility for your program.

1.2. Adding reference to the assembly

Now all that you need is to simply right-click on your project in Solution Explorer, then select „Add Reference..." from the pop-up menu.



Then on the Browse tab search for the SharpMidi.dll, select it, and push the OK button. Now you can access SharpMIDI through the SharpMidi namespace.

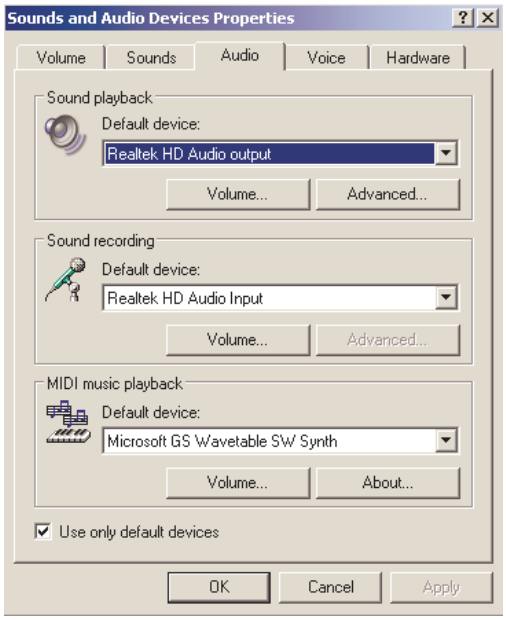
2. Quick example

After you referenced the SharpMidi.dll, described in the previous section, it's very easy to play some sound.

```
SharpMidi.Core.Init();                  //Initialize SharpMIDI

//Create proxy object
SharpMidi.Proxy p1 = new SharpMidi.Proxy();

p1.midiOutOpen(-1);                     //Open MS MIDI Mapper
p1.midiOutShortMsg(0x000000FF);         //Reset
p1.midiOutShortMsg(0x00000BC0);         //Select instrument
p1.midiOutShortMsg(0x00603C90);         //Note on

System.Threading.Thread.Sleep(1000); //Wait 1 second

p1.midiOutShortMsg(0x00003C80);         //Note off
```

This example will open the Microsoft MIDI Mapper, send some initialization messages to it, and play a sound. (See chapter 8. for more information on short midi messages.)

If you hear nothing (and your sound card is installed properly), then maybe you need to change the default MIDI device for Microsoft MIDI mapper. To do this, navigate here: *Control Panel > Sounds and Audio Devices > Audio*



Change the default device for MIDI music playback, then restart your program.

3. General usage

After added a reference to the assembly, the first thing you must do before using SharpMIDI is to initialize it with this static method:

```
SharpMidi.Core.Init();
```

But be careful, if you try to initialize it more then once the result will be an exception. After this, you may instantiate as many Proxy objects (maximum 64) as many MIDI devices you may want to use:

```
SharpMidi.Proxy p1 = new SharpMidi.Proxy();
SharpMidi.Proxy p2 = new SharpMidi.Proxy();
SharpMidi.Proxy p3 = new SharpMidi.Proxy();
```

Then using these proxies, you can get the number of MIDI output devices with the `int midiOutGetNumDevs()` method, or query the capabalities of a device with the `MidiOutCaps midiOutGetDevCaps(int device_id)` method, or do whatever you want.

After you finished using SharpMIDI, or you want to exit your application, you don't need to invoke any finalizer method or even don't need to close MIDI devices. They will close automatically, and all resources will be freed.

You can find full examples on usage at: http://bolner.hu/sharpmidi

4. The Proxy object

4.1. Overview

The proxy object delivers some Windows API functions to handle MIDI output devices, and to facilitate basic information query methods.

These functions are demonstrated in the forthcoming chapters, and their detailed reference can be found in chapter 11. The list of functions:

```
int midiOutGetNumDevs();
MidiOutCaps midiOutGetDevCaps(int device_id);
void midiOutOpen(int device_id);
void midiOutShortMsg(UInt32 msg);
void midiOutClose();
```

4.2. Istantiating proxy objects

It can be done as simple as this line:

```
SharpMidi.Proxy p1 = new SharpMidi.Proxy();
```

Keep in mind that there is an upper limit for proxy objects, and this limit is 64 (this is constant, and not related to the bitness of the operating system). If you exceed this limit, an exception occurs.

When you create a new proxy object, it forces the garbage collection, and waits until it ends, so you need only to delete all references to a proxy object to free up the device it may use. For example:

```
SharpMidi.Proxy p1, p2, p3;
p1 = new SharpMidi.Proxy();
p1.midiOutOpen(0);              //Open first MIDI output device
p1 = null;                      //Delete the one and only reference
p2 = new SharpMidi.Proxy();     //Waits for garbage collection
                                // before creating proxy object.
                                // Closing device in finalizer.
p2.midiOutOpen(0);              //No problem, opens device
p2.midiOutOpen(0);              //No problem, closes device 0 before
                                // opening it again.

p3 = new SharpMidi.Proxy();
p3.midiOutOpen(0);              //Second open attempt throws exception
```

4.3. Multithreading

Any number of threads can work on one proxy object at the same time, so the Proxy object is thread safe. Although be careful not to close a proxy's device connection while another thread uses that proxy to send messages.

4.4. Connection state

You can check any time the connection state of a proxy object, by its Status property:

```
StatusEnum Status
```

Where StatusEnum is: `public enum StatusEnum { Free, DeviceOpened }`
Free indicates that the proxy is not connected to a MIDI device, and DeviceOpened shows that there is a device opened by the proxy.

5. Getting the number of devices

In most cases the first thing to do (after the initialization) is to retrieve the number of MIDI output devices, with this function:

```
p1.midiOutGetNumDevs();
```

This returns an integer. If the returned number is 0, then the system does not contain any MIDI output devices. If the returned value is larger then 0, we have „n" number of devices, where n equal to the number returned by `midiOutGetNumDevs()`.

These devices can be identified by a zero based number, where the first device's ID is 0, the second's is 1 and so on. If the returned value is „n", then the ID of the devices:

0 First device
1 Second device
…
n-1 Last device

There is a special ID: -1. The number -1 identifies the Microsoft MIDI Mapper device, which is only a „port" for a real device. (See chapter 2.)

Numbering convention for devices (n is the number of devices):

-1 Microsoft MIDI Mapper
0 First device
1 Second device
…
n-1 Last device

One can use these IDs in the `midiOutGetDevCaps` and in the `midiOutOpen` function.

6. Device capabilities

6.1. Querying the capabilities of a MIDI output device

The query process can be simply done with this function:

```
MidiOutCaps midiOutGetDevCaps(int device_id)
```

The device ID is a zero based integer which we previously talked about in chapter 5 (dont forget that the number -1 is the ID of Microsoft MIDI Mapper). The returning MidiOutCaps contains information on the device.

6.2. MidiOutCaps

After querying the capabilities of a device with `midiOutGetDevCaps` it returns a `MidiOutCaps` object with the following fields:

Field	Type	Example
Name	string	Microsoft GS Wavetable SW Synth
ManufacturerCode	string	0x0001
ProductCode	string	0x0066
DriverVersion	string	5.10
Technology	TechnologyEnum	TechnologyEnum.SWSYNTH
TechnologyInEnglish	string	Software synthesizer
Voices	int	48
MaxNotes	int	48
ChannelMask	ushort	65535
ChannelMaskHEX	string	0xFFFF
Support	[Flags] SupportEnum	SupportEnum.CACHE | SupportEnum.LRVOLUME
SupportInEnglish	string	Supports patch caching; Supports separate left and right volume control

More information can be found on these fields at:
http://msdn.microsoft.com/en-us/library/ms711619(VS.85).aspx

6.3. Use in lists

The toString() method of MidiOutCaps returns the Name field, so you can simply add it to the items of a listBox.

```
int num = p1.midiOutGetNumDevs();
for (int i = -1; i < num; i++) {
        listBox1.Items.Add(p1.midiOutGetDevCaps(i));
}
```

7. How to open/close a midi device

Proxy objects are able to connect to MIDI devices by opening them. If a device is already opened by another proxy or by another program, an exception occurs. In that case one should close the other connection by closing the interfering program or the other proxy connection and try again.

Example of opening the first MIDI output device:

```
SharpMidi.Proxy p1 = new SharpMidi.Proxy();
p1.midiOutOpen(0);
```

(See chapter 5. for device numbering convention.)

If you have already opened a device by a proxy, and you use that proxy to open another device, `midiOutOpen` first closes the previous connection automatically, and opens the new:

```
p1.midiOutOpen(-1);          //Open Microsoft MIDI Mapper
```

If `midiOutOpen` fails to open a device (because it's busy), then it forces garbage collection first and tries again. It will repeat this process three times, and after the third unsuccessful attempt it throws an exception. Example:

```
SharpMidi.Proxy p1, p2;
p1 = new SharpMidi.Proxy();
p2 = new SharpMidi.Proxy();

p1.midiOutOpen(1);
p1 = null;
p2.midiOutOpen(1);         //At first midiOutOpen finds that the device
                           // is busy, so it forces garbage collection
                           // in the hope that it will free the device.
                           //And tries again with success (in this case).
```

A better way to close a device is to use the `midiOutClose` function:

```
p1.midiOutOpen(1);
p1.midiOutClose();
p2.midiOutOpen(1);
```

You can check the connection status of a proxy object, by its Status property:

```
if (p1.Status == SharpMidi.Proxy.StatusEnum.DeviceOpened)
{
    ...
}
```

See chapter 4.4. for more information.

8. Sending short midi messages

After connected to a MIDI output device, its very easy to send messages to it with the `midiOutShortMsg` function:

```
p1.midiOutShortMsg(0x00603C90);
```

A message is a 4 byte long unsigned integer.

9. How to construct short midi messages

As shown in the previous section, a short MIDI message is built-up of 4 bytes.

4th byte		3rd byte		2nd byte		1st byte	
HIGH	LOW	HIGH	LOW	HIGH	LOW	HIGH	LOW
0	0	6	0	3	C	9	0

The meaning of the bytes (or half bytes) in this example:

1st byte high: Note on
1st byte low: Channel 0 (0-15 where 10 is for percussion instruments)
2nd byte: Note number 0x3C = 60 (0-127)
3rd byte: Velocity 0x60 = 96 (0-127)
4th byte: *unused*

You can construct any message found in this summary:

http://www.midi.org/techspecs/midimessages.php

Some prerequisites needed to deal with thoose nasty messages:

Hexa numbers
http://en.wikipedia.org/wiki/Hexadecimal

Bitwise operators in C#:
http://www.blackwasp.co.uk/CSharpLogicalBitwiseOps.aspx

Bit masks:
http://www.vipan.com/htdocs/bitwisehelp.html

10. Exceptions

10.1. Overview

SharpMIDI contains unique exceptions to represent any error that can appeare on Windows API level. These exceptions are organized hierarchically (by inheritance) to make their handling more flexible.

You can find the exception hierarchy in chapter 10.3.

This manual distinguish between two kinds of exceptions. In the first group one can find exceptions which can be thrown by the Windows API functions. This group of exceptions is specified in chapter 11. All other exceptions can be found in chapter 10.4.

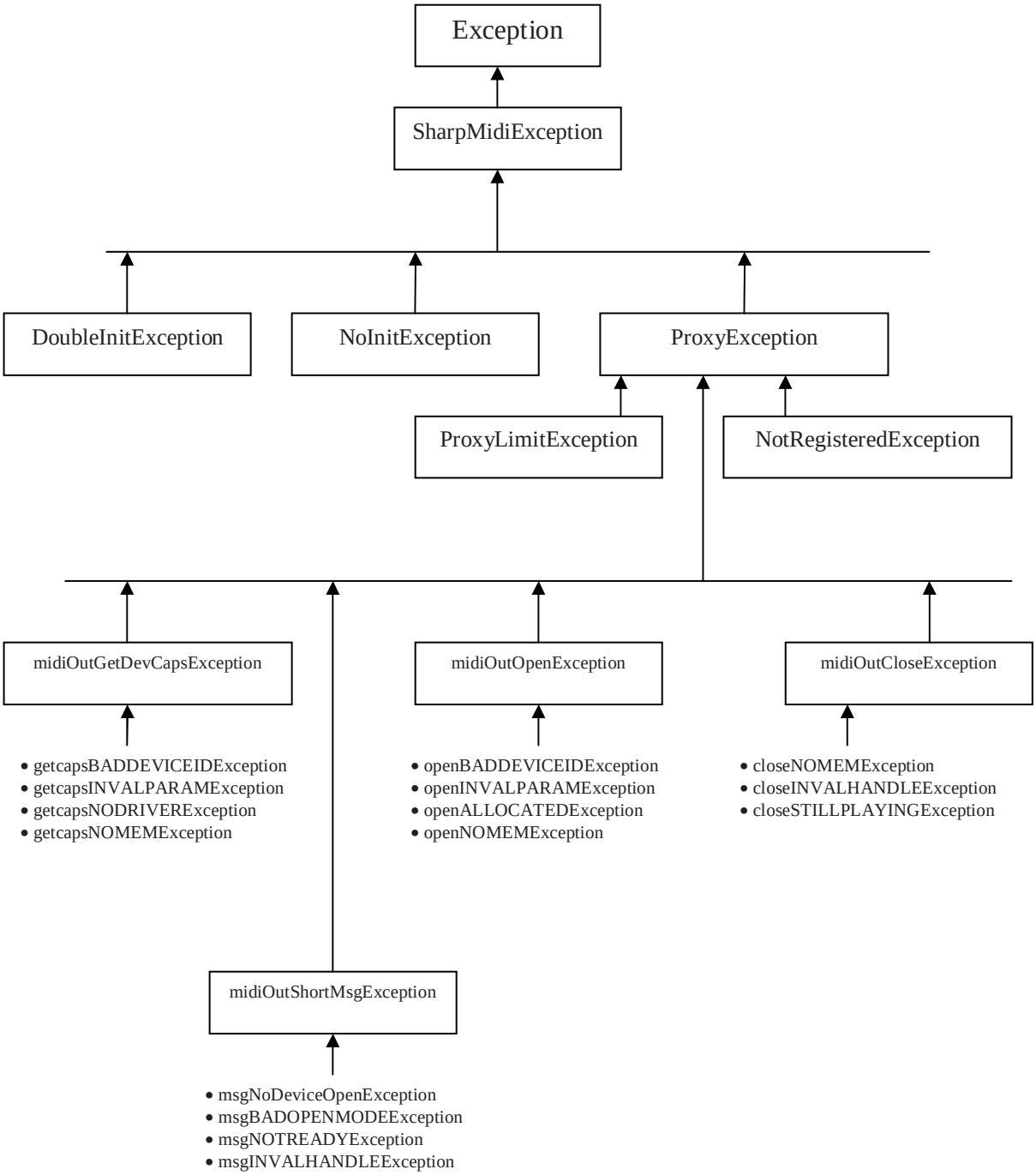
10.2. Catching exceptions

When you try to catch an exception, you first begin with specific ones and precede to the more and more general. If you have a look at the tree in next chapter, this movement from specific to general can be visualized as moving in the tree from the leaves to the root. (This tree is upside down.)

Example:

```
try
{
        ...Using SharpMIDI...
}
catch (SharpMidi.Exceptions.openBADDEVICEIDException ex)
{
        ...Handle exception...
}
catch (SharpMidi.Exceptions.midiOutOpenException ex)
{
        ...Handle exception...
}
catch (Exception ex)
{
        ...Handle exception...
}
```

10.3. Exception tree



10.4. Non Windows API related exceptions

- **DoubleInitException:** You tried to initialize SharpMIDI two times with the SharpMidi.Core.Init() method. Only one initialization is allowed.
- **NoInitException:** You tried to use a proxy object without the initialization of SharpMIDI with the SharpMidi.Core.Init() method. Initialization is the first thing to do.
- **NotRegisteredException:** To get this exception you must use the DeRegister() method of a Proxy object, and later try to use it. The DeRegister method is not documented because it's usage is not necessary.

11. Function reference

11.1. midiOutGetNumDevs

The midiOutGetNumDevs function retrieves the number of MIDI output devices present in the system.

```
int midiOutGetNumDevs()
```

Parameters:

- *none*

Return values:

- Returns the number of MIDI output devices. A return value of zero means that there are no devices (not that there is no error).

Exceptions:

- *no specific exception*

More information:
http://msdn.microsoft.com/en-us/library/ms711627(VS.85).aspx

11.2. midiOutGetDevCaps

The midiOutGetDevCaps function queries a specified MIDI output device to determine its capabilities.

```
MidiOutCaps midiOutGetDevCaps(int device_id)
```

Parameters:

- device_id: Identifier of the MIDI output device. The device identifier specified by this parameter varies from zero to one less than the number of devices present, or -1 for MIDI mapper.

Return values:

- Returns a MidiOutCaps object with information on the device. See chapter 6.2.

Exceptions:

- getcapsBADDEVICEIDException: The specified device identifier is out of range.
- getcapsINVALPARAMException: The specified pointer or structure is invalid.
- getcapsNODRIVERException: The driver is not installed.
- getcapsNOMEMException: The system is unable to load mapper string description.

More information:
http://msdn.microsoft.com/en-us/library/ms711621(VS.85).aspx

11.3. midiOutOpen

The midiOutOpen function opens a MIDI output device for playback..

```
void midiOutOpen(int device_id)
```

Parameters:

- device_id: Identifier of the MIDI output device. The device identifier specified by this parameter varies from zero to one less than the number of devices present, or -1 for MIDI mapper.

Return values:

- *none*

Exceptions:

- openBADDEVICEIDException: The specified device identifier is out of range.
- openINVALPARAMException: The specified pointer or structure is invalid.
- openALLOCATEDException: The specified resource is already allocated.
- openNOMEMException: The system is unable to allocate or lock memory.

(Or the general midiOutOpenException for other errors)

More information:
http://msdn.microsoft.com/en-us/library/ms711632(VS.85).aspx

11.4. midiOutShortMsg

The midiOutShortMsg function sends a short MIDI message to the specified MIDI output device.

```
void midiOutShortMsg(UInt32 msg)
```

Parameters:

- msg: MIDI message. The message is packed into an UInt32 value with the first byte of the message in the low-order byte. See chapter 8.

Return values:

- *none*

Exceptions:

- msgNoDeviceOpenException: You forget to open a device with the proxy object before using this function.
- msgBADOPENMODEException: The application sent a message without a status byte to a stream handle.
- msgNOTREADYException: The hardware is busy with other data.
- msgINVALHANDLEException: The specified device handle is invalid.

More information:
http://msdn.microsoft.com/en-us/library/ms711640(VS.85).aspx

11.5. midiOutClose

The midiOutClose function closes the MIDI output device connected to the proxy object.

```
void midiOutClose()
```

Parameters:

- *none*

Return values:

- *none*

Exceptions:

- closeNOMEMException: The system is unable to load mapper string description.
- closeINVALHANDLEException: The specified device handle is invalid.
- closeSTILLPLAYINGException: Buffers are still in the queue.

More information:
http://msdn.microsoft.com/en-us/library/ms711620(VS.85).aspx