How to use the win32midi package

Release 0.2

Soon Wah Chua

May 23, 2018

Abstract

This document describes how to use the win32midi package. It also describes the `player` that is part of the package. This is a work in progress.

Contents

Using the win32midi package is both fun and entertaining. You will be able to play or create music with the soundcard installed on your computer using the excellent Python language.

1 What is win32midi package?

It is a package to provide access to the MIDI functions available on Microsoft Windows platforms. It provides both low and high level access so that developers, musicians can decide which to use based on their needs.

2 Software Requirements

You need to have the following software installed:

- Python 2.7.x.
- Microsoft Win32 environment such as Windows 7,8,10.
- Latest Microsoft DirectX if your soundcard does not have a MIDI synthesizer.

Note that the `player` is provided with this package and implements some high level functions to play music in Python.

3 `win32midi` — Python wrapper functions for Win32 MIDI APIs

The `win32midi` module defines operations for calling the Win32 MIDI APIs via Python. Most of the functions will return an error code which can be decoded with `midiOutGetErrorTextA`. See the descriptions of the functions for details.

The `win32midi` module defines the following functions:

`midiInClose`(*hmi*)
: Closes the MIDI input device specified by the handle, *hmi*.

`midiInGetDevCapsA`(*dev,cap*)
: Queries the MIDI input device, *dev*, to determine its capablities, which are stored in the MIDIOUTCAPSA structure, *cap*. The result of the function call is returned.

`midiInGetErrorTextA`(*Error*)
: Returns the description text for the error code *Error* which is returned by most of the MIDI function calls in this module.

`midiInGetID`(*hmi*)
: Returns the device identifier for the MIDI input device specified by *hmi*.

`midiInGetNumDevs`()
: Returns the number of MIDI input devices.

`midiInOpen`(*DeviceID, Callback, CallbackInstance, Flags*)
: Open the MIDI device specified by the *DeviceID*. The *DeviceID* is a number from zero to one less than the value returned by midiInGetNumDevs. *Callback* is currently not implemented. It should be specified as 0 for no callback. CallbackInstance is currently not implemented and should be specified as 0. *Flags* should be specified as 0 for no callback mechanism. Other values of Flags are not implemented yet. This function returns a pair of numbers consisting the result of the call and a handle to the MIDI input device if call is successful.

`midiInPrepareHeader`(*hmi,hdr*)
: Prepares a MIDI system-exclusive or stream buffer, *hdr*, for input device specified by *hmi*.

`midiInReset`(*hmi*)
: Stops all the input on MIDI input device specified by *hmi*.

`midiInUnprepareHeader`(*hmi,hdr*)
: Cleans up the preparation performed by `midiInPrepareHeader` for input to device specified by *hmi*.

`midiOutClose`(*hmo*)
: Closes the MIDI output device specified by the handle, *hmo*.

`midiOutGetDevCapsA`(*dev,cap*)
: Queries the MIDI output device, *dev*, to determine its capablities, which are stored in the MIDIOUTCAPSA structure, *cap*. The result of the function call is returned.

`midiOutGetErrorTextA`(*Error*)
: Returns the description text for the error code *Error* which is returned by most of the MIDI function calls in this module.

`midiOutGetID`(*hmo*)
: Returns the device identifier for the MIDI output device specified by *hmo*.

`midiOutGetNumDevs`()
: Returns the number of MIDI output devices.

`midiOutGetVolume`(*hmo*)
: Returns a pair of numbers consisting of the result of the call and the current volume of a MIDI output device specified by the handle, *hmo*. The low-order word is the left-channel volume while the high-order word is the right channel setting. 0x0000 is silence. 0xFFFF is full volume.

`midiOutLongMsg`(*hmo,hdr*)
: Sends a system-exclusive MIDI message stored in the MIDIHDR structure, *hdr* to the MIDI output device specified by *hmo*. Result of the call is returned.

midiOutOpen (*DeviceID, Callback, CallbackInstance, Flags*)
Open the MIDI device specified by the *DeviceID*. The *DeviceID* is a number from zero to one less than the value returned by midiOutGetNumDevs. *Callback* is currently not implemented. It should be specified as 0 for no callback. CallbackInstance is currently not implemented and should be specified as 0. *Flags* should be specified as 0 for no callback mechanism. Other values of Flags are not implemented yet. This function returns a pair of numbers consisting the result of the call and a handle to the MIDI output device if call is successful.

midiOutPrepareHeader (*hmo,hdr*)
Prepares a MIDI system-exclusive or stream buffer, *hdr*, for output device specified by *hmo*.

midiOutReset (*hmo*)
Turns off all the notes on all MIDI channels for the MIDI output device specified by the handle, *hmo*.

midiOutSetVolume (*hmo,Volume*)
Set the current volume of a MIDI output device specified by the handle, *hmo* to the value, *Volume*. The low-order word of *Volume* is the left-channel volume while the high-order word is the right channel setting. 0x0000 is silence. 0xFFFF is full volume. The result of the call is returned.

midiOutShortMsg (*hmo, Msg*)
Sends a short MIDI message, *Msg*, to the MIDI output device specified by *hmo*.

midiOutUnprepareHeader (*hmo,hdr*)
Cleans up the preparation performed by `midiOutPrepareHeader` for output to device specified by *hmo*.

4 `player` — High level functions for playing music via the Win32 MIDI APIs

The `player` module defines simple high level APIs for playing music with the Windows MIDI functions via Python.

The `player` module defines the following features.

exception MIDIError
Exception raised when an operation fails for a specific reason. The exception argument is a string describing the reason of the failure.

class Player ([*dev=0*])
The Player class creates objects that provide the high level methods to play music in the API. Creates a Player object that is associated with MIDI device number *dev*.

The methods are listed below.

setInstrument (*self, num, chan=0*)
This method set the Instrument for playing music. It set the instrument used on the channel, *chan* to the instrument with GM MIDI patch number, *num*. The following table shows the instrument assigned in the GM MIDI with their patch number.

MIDI patch number	Instrument
	Piano
0	Acoustic grand piano
1	Bright acoustic piano
2	Electric grand piano
3	Honky-tonk piano
4	Rhodes piano
5	Chorused piano
6	Harpsichord
7	Clavinet
	Chromatic Percussion
8	Celesta
9	Glockenspiel

MIDI patch number	Instrument
10	Music box
11	Vibraphone
12	Marimba
13	Xylophone
14	Tubular bells
15	Dulcimer
	Organ
16	Hammond organ
17	Percussive organ
18	Rock organ
19	Church organ
20	Reed organ
21	Accordion
22	Harmonica
23	Tango accordion
	Guitar
24	Acoustic guitar (nylon)
25	Acoustic guitar (steel)
26	Electric guitar (jazz)
27	Electric guitar (clean)
28	Electric guitar (muted)
29	Overdriven guitar
30	Distortion guitar
31	Guitar harmonics
	Brass
32	Acoustic bass
33	Electric bass (finger)
34	Electric bass (pick)
35	Fretless bass
36	Slap bass 1
37	Slap bass 2
38	Synth bass 1
39	Synth bass 2
	Strings
40	Violin
41	Viola
42	Cello
43	Contrabass
44	Tremolo strings
45	Pizzicato strings
46	Orchestral harp
47	Timpani
	Ensemble
48	String ensemble 1
49	String ensemble 2
50	Synth. strings 1
51	Synth. strings 2
52	Choir Aahs
53	Voice Oohs
54	Synth voice
55	Orchestra hit
	Brass
56	Trumpet
57	Trombone
58	Tuba
59	Muted trumpet
60	French horn

MIDI patch number	Instrument
61	Brass section
62	Synth. brass 1
63	Synth. brass 2
	Reed
64	Soprano sax
65	Alto sax
66	Tenor sax
67	Baritone sax
68	Oboe
69	English horn
70	Bassoon
71	Clarinet
	Pipe
72	Piccolo
73	Flute
74	Recorder
75	Pan flute
76	Bottle blow
77	Shakuhachi
78	Whistle
79	Ocarina
	Synth Lead
80	Lead 1 (square)
81	Lead 2 (sawtooth)
82	Lead 3 (calliope lead)
83	Lead 4 (chiff lead)
84	Lead 5 (charang)
85	Lead 6 (voice)
86	Lead 7 (fifths)
87	Lead 8 (brass + lead)
	Synth Pad
88	Pad 1 (new age)
89	Pad 2 (warm)
90	Pad 3 (polysynth)
91	Pad 4 (choir)
92	Pad 5 (bowed)
93	Pad 6 (metallic)
94	Pad 7 (halo)
95	Pad 8 (sweep)
	Sound Effects
120	Guitar fret noise
121	Breath noise
122	Seashore
123	Bird tweet
124	Telephone ring
125	Helicopter
126	Applause
127	Gunshot

`setTempo`(*self, val*)
Sets the tempo of the music to be played to *val*. In other words, the music will be played to *val* beats per minute.

`setVolume`(*self, left, right*)
Sets the volume of the synthesizer to the values *left*, for left speaker, and *right*, for right speaker.

`getVolume`()

Gets the volumes of the synthesizer. The return value is a tuple, (left, right), for left and right speakers repectively.

`playNote`(*self, num, vel=120, len=0, chan=0*)

Plays the note, *num*, at velocity, *vel* in channel, *chan*. *num* is the MIDI note as defined in the MIDI specification, with Middle C at 60. To indicate a rest, use 99 for *num*. *vel* is the velocity of playing the note, the higher the louder the note is played. *chan* is the MIDI channel to play the note in.

`playChord`(*chord, vel=100, len=0, chan=0*)

Plays the chord, *chord*, at velocity, *vel* in channel, *chan*. *chord* is a list of MIDI notes, as defined in the MIDI specification, with Middle C at 60. To indicate a rest, use 99 for the value of the note. *vel* is the velocity of playing the note, the higher the louder the note is played. *chan* is the MIDI channel to play the chord in.

`playPiece`(*noteStruct,chordStruct*)

This method plays a piece of music defined with the melody notes in the *noteStruct* and the chord in the *chordStruct*. The *noteStruct* is a tuple with the first element to be a list of notes as defined in the table below and the second element to be a list of timing information. The *chordStruct* is defined similarly but each note is replaced by a list of notes that made up the chord to be played at that instance.

The following is a table that defines the notation to play the respective musical notes. The notation is designed as a mnemonic such that the first digit indicates the octave relative to the Middle C, while the second digit indicates the note within each octave with C as 1, D as 2 and so on. Sharps are indicated as half note higher, for example C# as 1.5

The table shows only notation defined for the definition of melody notes. For chord notes, the notes definitions are the same but each note is played at one octave lower. In other words, the C below Middle C is defined as 1 instead of -11. This is to reduce the number of typing for entering the chord notes.

Octave #	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-5	-51	-51.5	-52	-52.5	-53	-54	-55.5	-55	-55.5	-56	-56.5	-57
-4	-41	-41.5	-42	-42.5	-43	-44	-44.5	-45	-45.5	-46	-46.5	-47
-3	-31	-31.5	-32	-32.5	-33	-34	-34.5	-35	-35.5	-36	-36.5	-37
-2	-21	-21.5	-22	-22.5	-23	-24	-24.5	-25	-25.5	-26	-26.5	-27
-1	-11	-11.5	-12	-12.5	-13	-14	-14.5	-15	-15.5	-16	-16.5	-17
0	1	1.5	2	2.5	3	4	4.5	5	5.5	6	6.5	7
1	11	11.5	12	12.5	13	14	14.5	15	15.5	16	16.5	17
2	21	21.5	22	22.5	23	24	24.5	25	25.5	26	26.5	27
3	31	31.5	32	32.5	33	34	34.5	35	35.5	36	36.5	37
4	41	41.5	42	42.5	43	44	44.5	45	45.5	46	46.5	47
5	51	51.5	52	52.5	53	54	54.5	55				

For entering the timing information, the table below shows the simplified notation used for the different notes.

Note	**Notation**
Whole	1
Half	2
Quarter	4
Eighth	8
Sixteenth	16
32nd	32
64nd	64

For dotted notes, a .1 is appended to the notation. For example, a dotted quarter note will be indicated as 4.1

4.1 Example

Check out the files lullaby.py and aura_lee.py for examples of playing a piece of music.