関西国際大学研究紀要　第23号，2022年，297－304

（研究ノート）

Python ＋ Mido ライブラリによる MIDI データ処理機能の評価

Evaluation of MIDI data processing function by Python + Mido library

岡　本　　久*
Hisashi OKAMOTO

Abstract

This is a research note investigating the MIDI data processing function of the Mido library on Python. MIDI (Musical Instrument Digital Interface) is a standard for transmitting and receiving music performance information between electronic musical instruments and computers. Mido is a library that allows MIDI data to be handled as an object on the programming language Python. By using a library such as Mido, it can be expected to process the control and performance expression of various electronic musical instruments efficiently and flexibly. The author has made various attempts, such as converting information from various sensors into MIDI data based on certain rules, using a board computer such as Arduino or a personal computer. As a result of investigating the MIDI processing function of Mido this time, it was found to be sufficiently useful for the author's research. In the future, I would like to further explore the Python and Mido libraries and deepen my research activities.

キーワード：Python，Mido ライブラリ，MIDI

Ⅰ　はじめに

MIDI（Musical Instrument Digital Interface）とは，電子楽器やコンピュータ間で音楽の演奏情報などを送受信するための規格である。また Mido は，MIDI データをプログラム言語 Python 上のオブジェクトとして扱うことができるようにするためのライブラリである。Mido のようなライブラリを利用することで，様々な電子楽器との演奏情報などの送受信を効率よく，かつ柔軟に処理することが期待できる。Python 上の MIDI ライブラリはいくつか存在するが，今回は Mido ライブラリに着目しその機能を調べてみることとした。

*　関西国際大学　現代社会学部

Ⅱ．目的

著者はこれまで Arduino などのボードコンピュータやパソコンを使い，各種センサーからの入力情報などを一定のルールに基づき MIDI データに変換し，シンセサイザーを発音させるなど音楽の新たな演奏表現のためのヒューマンインターフェイスの試作や研究を行ってきた。しかしあらかじめ定めたルールに基づく分岐処理だけでは表現パターンが固定化されてしまうため，より柔軟かつ豊かな表現手法を試みるため，強力なデータ分析や学習機能を持つプログラム言語 Python を導入するとともに，Python 上での MIDI ライブラリの機能について確認することが本稿の目的である。

Ⅲ．Mido ライブラリによる Python プログラム

1．音を出すプログラム

表 1 は，Mido ライブラリによる音を出すプログラムである。わずか 5 行のプログラムで「ド」の音を出すことができる。むろんこれは Python が非常にシンプルにコーディングできるプログラム言語であるということが大きな理由であるが，いずれにせよこれだけの記述で音を出すことができるというのは，これまでのプログラミング言語では出来なかったであろう。また様々な種類がある MIDI メッセージは Message オブジェクトとして統一されており，どの MIDI メッセージなのかを調べるメソッドも存在するため（表 2），分岐処理などもシンプルな記述で行えそうである。

表 1 のプログラムは単に音をひとつ出すだけであり，またそのまま鳴りっぱなしになる。実用的なアプリケーションを作るとなれば相応のコーディング量を要するが，それでも Python のシンプルさはコーディングの量や時間を大幅に抑えることが期待できる。

表 1．Mido で「ド」の音を鳴らす Python プログラム

```
import mido
from mido import Message
outport = mido.open_output()
msg = Message('note_on', note=60)
outport.send(msg)
```

表 2．Message オブジェクトのメッセージの種類を調べる

```
import mido
from mido import Message
outport = mido.open_output()
msg1 = Message('note_on', note=60)
msg2 = Message('program_change', program=40)
print(msg1.type)
print(msg2.type)

――画面出力――
note_on
program_change
```

2．Mido ライブラリの MIDI 機能

多くのライブラリがそうであるように，Mido ライブラリのドキュメントはオンライン上で提供される HTML や PDF ファイルに限られ，それ以外は Web サイトに散在している様々な紹介事例なども拾い集めながらその機能を確認していくことになる。今回は特に正規の Mido ライブラリドキュメントで紹介されているサンプルを中心に，Mido の主な機能について調べてみた。

2．1．MIDI メッセージ

Modo のドキュメントによると，すべての MIDI メッセージに対応していると記されている。表 3 [1)] のように note on や note off，program_change といったチャンネル・メッセージはもとより，song_select などのコモンメッセージ，Start，Stop などのリアルタイムメッセージ，そして SysEx メッセージ（システム・エクスクルーシブ・メッセージ）にも対応していることが明記されている。機器の状態監視（ヘルスチェック）のための active_sensing メッセージについては，送信は可能であるが受信は行わないと記されているが，実用上あまり問題はないと考えられる。

表 3．Mido がサポートしている MIDI メッセージ

Name	Keyword Arguments / Attributes
note_off	channel note velocity
note_on	channel note velocity
Polytouch	channel note value
control_change	channel control value
program_change	channel program
Aftertouch	channel value
Pitchwheel	channel pitch
Sysex	Data
quarter_frame	frame_type frame_value
Songpos	Pos
song_select	Song
tune_request	
Clock	
Start	
Continue	
Stop	
active_sensing	
Reset	

2．2．チャンネル・メッセージ

チャンネル・メッセージは音の発音や停止，音色切り替えなど MIDI のもっとも基本的なメッセージであるが，そのオブジェクトは一行で生成することができる（表 4 ）。

表 4．MIDI メッセージ・オブジェクトの生成

```
msg1 = Message('note_on', channel=64, note=64)
msg2 = Message('note_on', channel=0, note=80, velocity=100)
```

また MIDI メッセージは［90H 40H 60H］のように HEX（16進数）コードで表記されるこ

とが多く，そのためのメソッドも用意されている（表５）。

表５．16進数による記述

```
msg1 = mido.Message.from_bytes([0x90, 0x40, 0x60])
msg2 = mido.Message.from_hex('90, 40 60')
```

その他コントロールチェンジやピッチベンダーなどのメッセージもすべて Message オブジェクトとして統一されているため，メッセージの種類ごとに異なるクラスやメソッドを用いる必要がなく，よりシンプルなコーディングが行える。ただしコントロールチェンジでの Bank Select や RPN（Registered Parameter Number），NRPN（Non-Registered Parameter Number）といった複数メッセージでひとつの機能を果たすものについては，一行で記述できるクラスやメソッドは提供されていない。今どきのシンセサイザーは数百以上の音色を有していることが普通なので，音色切り替えの際には Bank Select メッセージは常時使用することになる。具体的には表６のように Bank Select（MSB）＋ Bank Select（LSB）＋ Program Change の３つのメッセージによる音色指定が基本となる。このような複数メッセージで構成されるものについては，例えば表７のように１つの Message オブジェクトとして記述できないかと考える。むろん自らがそのようなクラスやメソッドを作ればよいだけのことではあるが，使用頻度の高いものについてはライブラリとして提供されることが望ましい。ただこの場合 outport.send（）メソッド内で複数メッセージを送信する処理が必要となり，メソッドとしての仕様が複雑化するため，別メソッドが必要になるであろう。

表６．Bank Select と Program Change による音色切り替えの方法

```
import mido
from mido import Message
outport = mido.open_output()
msg1 = Message('control_change', control=0, value=50)
msg2 = Message('control_change', control=32, value=13)
msg3 = Message('program_change', program=18)
outport.send(msg1)
outport.send(msg2)
outport.send(msg3)
```

表７．２つの Bank Select と Program Change を一行で記述

```
msg = Message('bankselect_program_change', bank_lsb=50, bank_msb=13, program=18)
outport.send(msg)
```

また Message オブジェクトは時間属性も設定できるため（表８），後述の SMF トラックのデータを扱う際にも問題なくプログラムを組むことができる。

表８．Message オブジェクトの時間属性設定

```
msg = Message('note_on', channel=0, note=48, velocity=64, time=2000)
```

2．3．SysEx メッセージ

Mido では SysEx メッセージ（System Exclusive Message）も扱えるため，様々な機器とのパッチデータの送受信なども問題なく行える。また SysEx メッセージ専用の SYX ファイルの読み書きも行えるため，機器の初期設定値やダンプデータを個別ファイルとして管理できる。さらにバイナリ形式だけでなく，テキスト形式での読み書きの機能も提供されているため，テキストエディタで SysEx メッセージの確認や編集が行えるなど利便性がある。

ただし SysEx メッセージの多くは，データの最後尾（EOX の直前）にチェックサム値を入れる仕様であるため，データ内容が変更されればこの値を再計算する必要がある。Mido にはこのチェックサム値計算のためのメソッドは提供されていないため，Python 上の別のライブラリを利用するか，独自メソッドを作成する必要がある。（表 9 ）

表 9．SysEx メッセージのデータ構造

F0H	・・・・可変長データ・・・・	Checksum	F7H（EOX）

2．4．MIDI ポート

Mido ポートは，MIDI メッセージを送受信するためのクラス（python-rtmidi ライブラリの追加インストールが必要）で，たとえばキーボード（鍵盤装置）からの演奏情報の受信や，シンセサイザーなどへのデータ送信のための入出力ポートを提供する（表10）。シンセサイザーや Audio/MIDI インターフェイスなどの機器が接続されていないパソコンの場合，送信ポートは Microsoft GS Wavetable Synth が音源として認識される（Windows の場合）。鍵盤装置やシンセサイザーなどが接続されている場合は，それに応じて入力・出力ポートが認識され，特定のポートを介してデータの入出力が行える。

また入力・出力ともに複数ポート（MultiPort）への入出力もサポートされているので，２台の鍵盤の演奏データの同時受信や，複数シンセサイザーへのデータ送信の振り分けなども行える。さらにコールバック関数も提供されているため，受信処理をループで待ち受ける必要がなく，他の処理に専念できる。

表10．MIDI ポートの入出力設定およびポート情報の取得

```
import mido
from mido import Message
outport = mido.open_output()
inport = mido.open_input()
print(outport)
print(inport)

――画面出力――
<open output 'Microsoft GS Wavetable Synth 0' (RtMidi/WINDOWS_MM)>
<open input 'microKEY2 1 KEYBOARD 0' (RtMidi/WINDOWS_MM)>
```

2．5．MIDI ファイル

MIDI ファイルについては，SMF フォーマット（Standard MIDI File／標準 MIDI ファイル）

に対応しており，ファイルの読み込み・書き込み・再生，そして各トラックやトラック内データの追加や削除などすべての操作が行える。表11は song.mid ファイルを読み込み，再生するプログラムである。

なお MusicXML ファイルについては記述がないため，対応していないようである。むろん MusicXML は主に楽譜ソフト間でのレイアウト情報に関する共通規格なので，Mido ライブラリとして必ずしもサポートする必要はないと思われる。また MusicXML を扱うのであれば，たとえば MIT メディアラボによる music21ライブラリがあるため，これを利用すれば MusicXML の編集や，フリーの楽譜ソフト MuseScore への楽譜データの受け渡しを行うことができるため，Mido と music21のライブラリを組み合わせることで，幅広く音楽情報データに対応することができる。

表11．SMF ファイルの読み込みと再生

```
import mido
from mido import MidiFile
outport = mido.open_output()
for msg in MidiFile('song.mid').play():
    outport.send(msg)
```

2．6．MIDI トラック編集機能

Mido には SMF トラックの操作や，各トラック内のデータ編集のための様々なメソッドが用意されている。メタメッセージやテンポチェンジ，４分音符あたりのティック数の設定など SMF の様々なデータ操作についても問題なく操作できる。編集機能の応用として，たとえばひとつの楽曲に対し，複数の伴奏やドラムのパターンを用意し，必要に応じトラックを入れ替えるなどすれば様々なアレンジ曲として提供することができる。表12は original.mid ファイル（track［１］＝旋律パート／track［２］＝伴奏パート）を，あらかじめ用意した別の伴奏パターン（pattern_a.mid）に入れ替えるプログラムである。また事前にパターンを用意しておくだけでなく，Python の学習機能などを使い新たなパターンを自動生成するなど，様々な展開が考えられる。

表12．伴奏パターンを入れ替えるプログラム

```
import mido
from mido import MidiFile
mid1 = MidiFile('original.mid', clip=True)
mid2 = MidiFile('pattern_a.mid', clip=True)
mid1.tracks.remove(tracks[2])
mid1.tracks.append(mid2.tracks[1])
mid1.save('arrange.mid')
```

2．7．MIDI メッセージの解析

Mido は MIDI メッセージを解析し，Message オブジェクトに変換するパーサー（構文解析）機能も備えている（表13）。MIDI データの受信では，ランニングステータスやリアルタイムメッセージの処理など面倒なことが多いため，このようなパーサー機能が存在することは大変ありがたい。

表13. Mido の MIDI メッセージ解析機能

```
import mido
print(mido.parse([0x90, 0x40, 0x40]))
print(mido.parse_all([0xC0, 0x53, 0x93, 0x48, 0x40, 0x83, 0x48, 0x00]))

――画面出力――
note_on channel=0 note=64 velocity=64 time=0
[Message('program_change', channel=0, program=83, time=0), Message('note_on', channel=3,
note=72, velocity=64, time=0), Message('note_off', channel=3, note=72, velocity=0, time=0)]
```

この他にも，MIDI メッセージをテキスト形式に変換（および逆変換）するメソッドも提供されている。バイナリ形式である MIDI データをテキストとして扱えるようになれば，専用の編集ソフトを用いることなく，テキストエディタだけで編集を行うことができる。

2．8．ソケットポート

Mido には MIDIoverTCP／IP という，特定の TCP／IP ポートを通じて MIDI メッセージを送受信する機能が提供されている。この機能はまだ試していないが，異なるコンピュータのアプリケーション間でのデータ通信や同期が行える。現在多くの音楽ソフトでは Rewire という複数のアプリケーション間でオーディオや MIDI データをリアルタイムに送受信するシステムが利用できるようになっているが，ネットワーク間での送受信に対応した MIDIoverTCP／IP は今後様々な活用に期待が持てる。

Ⅳ．まとめ ― Python ＋ Mido による MIDI データ処理の有用性

今回簡単にではあるが，ひととおり Mido ライブラリによる MIDI データ処理機能を調べてみた。結果としては，MIDI の各種データ処理のためのクラスやメソッドが必要十分に備えられているため，大変有用なライブラリであると判断できる。これだけの機能があれば，Cubase や Studio One などの DAW（Digital Audio Workstation）のようなソフト開発を Python 上で行うことも十分可能であろう。

2020年 2 月，一般財団法人音楽電子事業協会（AMEI）のホームページ上で，MIDI2.0企画書が公開された。概要ではこれまでの MIDI1.0との互換性を保ちつつ，新たな2.0通信プロトコル以外にも MIDI 機器間ネゴシエーションとして，機器間でのプロファイルの共通化，機器の持つプロパティを取得／設定などの規格が提示されている。MIDI の規格は1982年に作られ，これまでに MIDI 規格推奨応用事例（RP）として様々な改定はされてきたが，長い年月に渡り基本は version 1.0のままであった。すでに様々なメーカーが独自の規格を定め運用している中，ようやくというか今さらといった感は拭えないが，待ち望まれた改訂であり今後の普及発展に期待したい。

今回 Python ＋ Mido ライブラリについて調べてみて感じたことは，MIDI プログラムが非常にシンプルかつ効率よくできるようになっているということである。今後さらに研究を進め，様々音楽表現の可能性を探っていきたい。

1）Mido - MIDI Objects for Python（Message Type）より転載,
https://mido.readthedocs.io/en/latest/message_types.html（最終確認日 2021/09/17）

【参考文献】

・Kazuhiro Kobayashi- Mido MIDI Objects for Python,
https://kaz-kobayashi.github.io/mido/（最終確認日 2021/09/10）

・Mido - MIDI Objects for Python (Version 1.2.10),
https://mido.readthedocs.io/en/latest/（最終確認日 2021/09/10）

・Python 3.9.7 documentation,
https://docs.python.org/3/（最終確認日 2021/09/10）

・音楽電子事業協会「MIDI1.0規格書」PDF ファイル,
http://amei.or.jp/midistandardcommittee/MIDIspcj.html（最終確認日 2021/09/10）

・音楽電子事業協会「MIDI2.0規格書（英語版）」PDF ファイル,
http://amei.or.jp/midistandardcommittee/MIDI2.0/MIDIspcj2.html（最終確認日 2021/09/10）