MusPy

Nov 06, 2020

Contents

MusPy is an open source Python library for symbolic music generation. It provides essential tools for developing a music generation system, including dataset management, data I/O, data preprocessing and model evaluation.

CHAPTER 1

Features

- Dataset management system for commonly used datasets with interfaces to PyTorch and TensorFlow.
- Data I/O for common symbolic music formats (e.g., MIDI, MusicXML and ABC) and interfaces to other symbolic music libraries (e.g., music21, mido, pretty_midi and Pypianoroll).
- Implementations of common music representations for music generation, including the pitch-based, the event-based, the piano-roll and the note-based representations.
- Model evaluation tools for music generation systems, including audio rendering, score and piano-roll visualizations and objective metrics.

Here is an overview of the library.

CHAPTER 2

Why MusPy

A music generation pipeline usually consists of several steps: data collection, data preprocessing, model creation, model training and model evaluation.

While some components need to be customized for each model, others can be shared across systems. For symbolic music generation in particular, a number of datasets, representations and metrics have been proposed in the literature. As a result, an easy-to-use toolkit that implements standard versions of such routines could save a great deal of time and effort and might lead to increased reproducibility.

CHAPTER 3

Installation

To install MusPy, please run `pip install muspy`. To build MusPy from source, please download the source and run `python setup.py install`.

CHAPTER 4

Documentation

Documentation is available here and as docstrings with the code.

CHAPTER 5

Citing

Please cite the following paper if you use MusPy in a published work:

Hao-Wen Dong, Ke Chen, Julian McAuley, and Taylor Berg-Kirkpatrick, "MusPy: A Toolkit for Symbolic Music Generation," in *Proceedings of the 21st International Society for Music Information Retrieval Conference (ISMIR)*, 2020.

[homepage] [video] [paper] [slides] [poster] [arXiv] [code] [documentation]

CHAPTER 6

Disclaimer

This is a utility library that downloads and prepares public datasets. We do not host or distribute these datasets, vouch for their quality or fairness, or claim that you have license to use the dataset. It is your responsibility to determine whether you have permission to use the dataset under the dataset's license.

If you're a dataset owner and wish to update any part of it (description, citation, etc.), or do not want your dataset to be included in this library, please get in touch through a GitHub issue. Thanks for your contribution to the community!

CHAPTER 7

Contents

7.1 Getting Started

Welcome to MusPy! We will go through some basic concepts in this tutorial.

Hint: Be sure you have MusPy installed. To install MusPy, please run `pip install muspy`.

In the following example, we will use this JSON file as an example.

First of all, let's import the MusPy library.

```
import muspy
```

Now, let's load the JSON file into a Music object.

```
music = muspy.load("example.json")
print(music)
```

Here's what we got.

```
Music(metadata=Metadata(schema_version='0.0', title='Für Elise', creators=['Ludwig␣
↪van Beethoven'], collection='Example dataset', source_filename='example.json'),␣
↪resolution=4, tempos=[Tempo(time=0, qpm=72.0)], key_signatures=[KeySignature(time=0,
↪ root=9, mode='minor')], time_signatures=[TimeSignature(time=0, numerator=3,␣
↪denominator=8)], downbeats=[4, 16], lyrics=[Lyric(time=0, lyric='Nothing but a lyric
↪')], annotations=[Annotation(time=0, annotation='Nothing but an annotation')],␣
↪tracks=[Track(program=0, is_drum=False, name='Melody', notes=[Note(time=0,␣
↪duration=2, pitch=76, velocity=64), Note(time=2, duration=2, pitch=75, velocity=64),
↪ Note(time=4, duration=2, pitch=76, velocity=64), ...], lyrics=[Lyric(time=0, lyric=
↪'Nothing but a lyric')], annotations=[Annotation(time=0, annotation='Nothing but an␣
↪annotation')])])
```

Hard to read, isn't it? Let's print it beautifully.

```
music.print()
```

Now here's what we got.

```
metadata:
  schema_version: '0.0'
  title: Für Elise
  creators: [Ludwig van Beethoven]
  collection: Example dataset
  source_filename: example.json
resolution: 4
tempos:
  - {time: 0, qpm: 72.0}
key_signatures:
  - {time: 0, root: 9, mode: minor}
time_signatures:
  - {time: 0, numerator: 3, denominator: 8}
downbeats: [4, 16]
lyrics:
  - {time: 0, lyric: Nothing but a lyric}
annotations:
  - {time: 0, annotation: Nothing but an annotation}
tracks:
  - program: 0
    is_drum: false
    name: Melody
    notes:
      - {time: 0, duration: 2, pitch: 76, velocity: 64}
      - {time: 2, duration: 2, pitch: 75, velocity: 64}
      - {time: 4, duration: 2, pitch: 76, velocity: 64}
      - {time: 6, duration: 2, pitch: 75, velocity: 64}
      - {time: 8, duration: 2, pitch: 76, velocity: 64}
      - {time: 10, duration: 2, pitch: 71, velocity: 64}
      - {time: 12, duration: 2, pitch: 74, velocity: 64}
      - {time: 14, duration: 2, pitch: 72, velocity: 64}
      - {time: 16, duration: 2, pitch: 69, velocity: 64}
    lyrics:
      - {time: 0, lyric: Nothing but a lyric}
    annotations:
      - {time: 0, annotation: Nothing but an annotation}
```

You can use dot notation to assess the data. For example, `music.metadata.title` returns the song title, and `music.tempos[0].qpm` returns the first tempo in qpm (quarter notes per minute). If you want a list of all the pitches, you can do

```
print([note.pitch for note in music.tracks[0].notes])
```

Then you will get `[76, 75, 76, 75, 76, 71, 74, 72, 69]`.

Hint: `music[i]` is a shorthand for `music.tracks[i]`, and `len(music)` for `len(music.tracks)`.

There's more MusPy offers. Here is an example of data preparation pipeline using MusPy.

And here is another example of result writing pipeline using MusPy.

7.2 MusPy Classes

MusPy provides several classes for working with symbolic music. Here is an illustration of the relations between different MusPy classes.

7.2.1 Base Classes

Base Class

All MusPy classes inherit from the *muspy.Base* class. A *muspy.Base* object supports the following operations.

- *muspy.Base.to_ordered_dict()*: convert the content into an ordered dictionary
- *muspy.Base.from_dict()* (class method): create a MusPy object of a certain class
- *muspy.Base.print()*: show the content in a YAML-like format
- *muspy.Base.validate()*: validate the data stored in an object
- *muspy.Base.is_valid()*: return a boolean indicating if the stored data is valid
- *muspy.Base.adjust_time()*: adjust the timing of an object

ComplexBase Class

MusPy classes that contains list attributes also inherit from the *muspy.ComplexBase* class. A *muspy.ComplexBase* object supports the following operations.

- *muspy.ComplexBase.append()*: append an object to the corresponding list
- *muspy.ComplexBase.remove_invalid()*: remove invalid items from the lists
- *muspy.ComplexBase.sort()*: sort the lists
- *muspy.ComplexBase.remove_duplicate()*: remove duplicate items from the lists

7.2.2 Music Class

The *muspy.Music* class is the core element of MusPy. It is a universal container for symbolic music.

Attributes	Description	Type	Default
metadata	Metadata	*muspy.Metadata*	*muspy.Metadata()*
resolution	Time steps per beat	int	`muspy.DEFAULT_RESOLUTION`
tempos	Tempo changes	list of *muspy.Tempo*	[]
key_signatures	Key signature changes	list of *muspy.KeySignature*	[]
time_signatures	Time signature changes	list of *muspy.TimeSignature*	[]
downbeats	Downbeat positions	list of int	[]
lyrics	Lyrics	list of *muspy.Lyric*	[]
annotations	Annotations	list of *muspy.Annotation*	[]
tracks	Music tracks	list of *muspy.Track*	[]

Hint: An example of a MusPy Music object as a YAML file is available here.

class muspy.**Music**(*metadata: Optional[muspy.classes.Metadata] = None*, *resolution: Optional[int] = None*, *tempos: Optional[List[muspy.classes.Tempo]] = None*, *key_signatures: Optional[List[muspy.classes.KeySignature]] = None*, *time_signatures: Optional[List[muspy.classes.TimeSignature]] = None*, *downbeats: Optional[List[int]] = None*, *lyrics: Optional[List[muspy.classes.Lyric]] = None*, *annotations: Optional[List[muspy.classes.Annotation]] = None*, *tracks: Optional[List[muspy.classes.Track]] = None*)

A universal container for symbolic music.

This is the core class of MusPy. A Music object can be constructed in the following ways.

- *muspy.Music()*: Construct by setting values for attributes.
- muspy.Music.from_dict(): Construct from a dictionary that stores the attributes and their values as key-value pairs.
- *muspy.read()*: Read from a MIDI, a MusicXML or an ABC file.
- *muspy.load()*: Load from a JSON or a YAML file saved by *muspy.save()*.
- *muspy.from_object()*: Convert from a *music21.Stream*, a mido.MidiFile, a pretty_midi.PrettyMIDI or a pypianoroll.Multitrack object.

metadata
Metadata.

Type *muspy.Metadata*

resolution
Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Type int, optional

tempos
Tempo changes.

Type list of *muspy.Tempo*

key_signatures
Key signatures changes.

Type list of *muspy.KeySignature*

time_signatures
Time signature changes.

Type list of *muspy.TimeSignature*

downbeats
Downbeat positions.

Type list of int

lyrics
Lyrics.

Type list of *muspy.Lyric*

annotations
Annotations.

Type list of *muspy.Annotation*

tracks
Music tracks.

Type list of *muspy.Track*

Note: Indexing a Music object returns the track of a certain index. That is, `music[idx]` returns `music.tracks[idx]`. Length of a Music object is the number of tracks. That is, `len(music)` returns `len(music.tracks)`.

---

adjust_resolution(*target: Optional[int] = None*, *factor: Optional[float] = None*, *rounding: Union[str, Callable, None] = 'round'*) → muspy.music.Music
Adjust resolution and timing of all time-stamped objects.

Parameters

- **target** (*int, optional*) – Target resolution.
- **factor** (*int or float, optional*) – Factor used to adjust the resolution based on the formula: *new_resolution = old_resolution * factor*. For example, a factor of 2 double the resolution, and a factor of 0.5 halve the resolution.
- **rounding** (*{'round', 'ceil', 'floor'} or callable, optional*) – Rounding mode. Defaults to 'round'.

Returns

Return type Object itself.

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType
Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

append(*obj*) → ComplexBaseType
Append an object to the correseponding list.

This will automatically determine the list attributes to append based on the type of the object.

Parameters **obj** – Object to append.

clip(*lower: int = 0*, *upper: int = 127*) → muspy.music.Music
Clip the velocity of each note for each track.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType
Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

get_end_time(*is_sorted: bool = False*) → int
Return the the time of the last event in all tracks.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations.

Parameters **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

get_real_end_time(*is_sorted: bool = False*) → float
Return the end time in realtime.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations. Assume 120 qpm (quarter notes per minute) if no tempo information is available.

Parameters **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

remove_duplicate(*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType
Remove duplicate items from a list attribute.

Parameters

- **attr** (*str*) – Attribute to check. Defaults to check all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

remove_invalid(*attr: Optional[str] = None, recursive: bool = True*) → ComplexBaseType
Remove invalid items from a list attribute.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

save(*path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs*)
Save loselessly to a JSON or a YAML file.

Refer to *muspy.save()* for full documentation.

save_json(*path: Union[str, pathlib.Path], **kwargs*)
Save loselessly to a JSON file.

Refer to *muspy.save_json()* for full documentation.

save_yaml(*path: Union[str, pathlib.Path]*)
Save loselessly to a YAML file.

Refer to *muspy.save_yaml()* for full documentation.

show(*kind: str*, ***kwargs*)
Show visualization.

Refer to *muspy.show()* for full documentation.

show_pianoroll(***kwargs*)
Show pianoroll visualization.

Refer to *muspy.show_pianoroll()* for full documentation.

show_score(***kwargs*)
Show score visualization.

Refer to *muspy.show_score()* for full documentation.

sort(*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Sort a list attribute.

Parameters

- **attr** (*str*) – Attribute to sort. Defaults to sort all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

synthesize(***kwargs*) → numpy.ndarray
Synthesize a Music object to raw audio.

Refer to *muspy.synthesize()* for full documentation.

to_event_representation(***kwargs*) → numpy.ndarray
Return in event-based representation.

Refer to *muspy.to_event_representation()* for full documentation.

to_mido(***kwargs*) → mido.midifiles.midifiles.MidiFile
Return as a MidiFile object.

Refer to *muspy.to_mido()* for full documentation.

to_music21(***kwargs*) → music21.stream.Stream
Return as a Stream object.

Refer to *muspy.to_music21()* for full documentation.

to_note_representation(***kwargs*) → numpy.ndarray
Return in note-based representation.

Refer to *muspy.to_note_representation()* for full documentation.

to_object(*kind: str*, ***kwargs*)
Return as an object in other libraries.

Refer to *muspy.to_object()* for full documentation.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

to_pianoroll_representation(***kwargs*) → numpy.ndarray
Return in piano-roll representation.

Refer to *muspy.to_pianoroll_representation()* for full documentation.

to_pitch_representation(***kwargs*) → numpy.ndarray
Return in pitch-based representation.

Refer to *muspy.to_pitch_representation()* for full documentation.

to_pretty_midi(***kwargs*) → pretty_midi.pretty_midi.PrettyMIDI
Return as a PrettyMIDI object.

Refer to *muspy.to_pretty_midi()* for full documentation.

to_pypianoroll(***kwargs*) → pypianoroll.multitrack.Multitrack
Return as a Multitrack object.

Refer to *muspy.to_pypianoroll()* for full documentation.

to_representation(*kind: str*, ***kwargs*) → numpy.ndarray
Return in a specific representation.

Refer to *muspy.to_representation()* for full documentation.

transpose(*semitone: int*) → muspy.music.Music
Transpose all the notes by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

Notes

Drum tracks are skipped.

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

write(*path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs*)
Write to a MIDI, a MusicXML, an ABC or an audio file.

Refer to *muspy.write()* for full documentation.

write_abc(*path: Union[str, pathlib.Path], **kwargs*)
Write to an ABC file.

Refer to *muspy.write_abc()* for full documentation.

write_audio(*path: Union[str, pathlib.Path], **kwargs*)
Write to an audio file.

Refer to *muspy.write_audio()* for full documentation.

write_midi(*path: Union[str, pathlib.Path], **kwargs*)
Write to a MIDI file.

Refer to *muspy.write_midi()* for full documentation.

write_musicxml(*path: Union[str, pathlib.Path], **kwargs*)
Write to a MusicXML file.

Refer to *muspy.write_musicxml()* for full documentation.

7.2.3 Track Class

The *muspy.Track* class is a container for music tracks. In MusPy, each track contains only one instrument.

Attributes	Description	Type	Default
program	MIDI program number	int (0-127)	0
is_drum	If it is a drum track	bool	False
name	Track name	str	
notes	Musical notes	list of *muspy.Note*	[]
chords	Chords	list of *muspy.Chord*	[]
lyrics	Lyrics	list of *muspy.Lyric*	[]
annotations	Annotations	list of *muspy.Annotation*	[]

(MIDI program number is based on General MIDI specification; see here.)

class muspy.**Track**(*program: int = 0, is_drum: bool = False, name: Optional[str] = None, notes: Optional[List[muspy.classes.Note]] = None, chords: Optional[List[muspy.classes.Chord]] = None, lyrics: Optional[List[muspy.classes.Lyric]] = None, annotations: Optional[List[muspy.classes.Annotation]] = None*)

A container for music track.

program
Program number according to General MIDI specification[1]. Defaults to 0 (Acoustic Grand Piano).

Type int, 0-127, optional

is_drum
Whether it is a percussion track. Defaults to False.

Type bool, optional

name
Track name.

Type str, optional

notes
Musical notes. Defaults to an empty list.

Type list of *muspy.Note*, optional

chords
Chords. Defaults to an empty list.

Type list of *muspy.Chord*, optional

annotations
Annotations. Defaults to an empty list.

Type list of *muspy.Annotation*, optional

lyrics
Lyrics. Defaults to an empty list.

Type list of *muspy.Lyric*, optional

Note: Indexing a Track object returns the note at a certain index. That is, `track[idx]` returns `track.notes[idx]`. Length of a Track object is the number of notes. That is, `len(track)` returns `len(track.notes)`.

References

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType
Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

Returns

Return type Object itself.

append(*obj*) → ComplexBaseType
Append an object to the correseponding list.

This will automatically determine the list attributes to append based on the type of the object.

Parameters **obj** – Object to append.

clip(*lower: int = 0*, *upper: int = 127*) → muspy.classes.Track
Clip the velocity of each note.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType
Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

get_end_time(*is_sorted: bool = False*) → int
Return the time of the last event.

This includes notes, chords, lyrics and annotations.

Parameters **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

remove_duplicate(*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Remove duplicate items from a list attribute.

Parameters

- **attr** (*str*) – Attribute to check. Defaults to check all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

remove_invalid(*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Remove invalid items from a list attribute.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

sort (*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Sort a list attribute.

Parameters

- **attr** (*str*) – Attribute to sort. Defaults to sort all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

to_ordered_dict (*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

transpose (*semitone: int*) → muspy.classes.Track
Transpose the notes by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

validate (*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type (*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.4 Metadata Class

The *muspy.Metadata* class is a container for metadata.

Attributes	Description	Type	Default
schema_version	Schema version	str	'0.0'
title	Song title	str	
creators	Creators(s) of the song	list of str	[]
copyright	Copyright notice	str	
collection	Name of the collection	str	
source_filename	Name of the source file	str	
source_format	Format of the source file	str	

class muspy.**Metadata**(*schema_version: str = '0.0', title: Optional[str] = None, creators: Optional[List[str]] = None, copyright: Optional[str] = None, collection: Optional[str] = None, source_filename: Optional[str] = None, source_format: Optional[str] = None*)

A container for metadata.

schema_version
Schema version. Defaults to the latest version.

Type str

title
Song title.

Type str, optional

creators
Creator(s) of the song.

Type list of str, optional

copyright
Copyright notice.

Type str, optional

collection
Name of the collection.

Type str, optional

source_filename
Name of the source file.

Type str, optional

source_format
Format of the source file.

Type str, optional

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType

Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.5 Tempo Class

The *muspy.Tempo* class is a container for tempos.

Attributes	Description	Type	Default
time	Start time of the tempo	int	
qpm	Tempo in qpm (quarter notes per minute)	float	

class muspy.**Tempo**(*time: int*, *qpm: float*)
A container for key signature.

time
Start time of the tempo, in time steps.

Type int

qpm
Tempo in qpm (quarters per minute).

Type float

adjust_time(*func: Callable[[int], int]*, *attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType
Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.6 KeySignature Class

The *muspy.KeySignature* class is a container for key signatures.

Attributes	Description	Type	Default
time	Start time	int	
root	Root note as a number	int	
mode	Mode (e.g., "major")	str	
root_str	Root note as a string	int	

class muspy.**KeySignature**(*time: int*, *root: Optional[int] = None*, *mode: Optional[str] = None*, *fifths: Optional[int] = None*, *root_str: Optional[str] = None*)

A container for key signature.

time
Start time of the key signature, in time steps or seconds.

Type int

root
Root of the key signature.

Type int, optional

mode
Mode of the key signature.

Type str, optional

fifths
Number of flats or sharps. Positive numbers for sharps and negative numbers for flats.

Type int, optional

root_str
Root of the key signature as a string.

Type str, optional

adjust_time(*func: Callable[[int], int]*, *attr: Optional[str] = None*, *recursive: bool = True*) → BaseType

Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.7 TimeSignature Class

The *muspy.TimeSignature* class is a container for time signatures.

Attributes	Description	Type	Default
time	Start time	int	
numerator	Numerator (e.g., “3” for 3/4)	int	
denominator	Denominator (e.g., “4” for 3/4)	int	

class muspy.**TimeSignature**(*time: int*, *numerator: int*, *denominator: int*)
A container for time signature.

time
Start time of the time signature, in time steps or seconds.

Type int

numerator
Numerator of the time signature.

Type int

denominator
Denominator of the time signature.

Type int

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType
Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType
Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{“attr1”: value1, “attr2”: value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute’s attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.8 Lyric Class

The `muspy.` class is a container for lyrics.

Attributes	Description	Type	Default
time	Start time	int	
lyric	Lyric (sentence, word, syllable, etc.)	str	

class muspy.**Lyric**(*time: int*, *lyric: str*)
A container for lyric.

time
Start time of the lyric, in time steps or seconds.

Type int

lyric
Lyric (sentence, word, syllable, etc.).

Type str

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType

Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None, recursive: bool = True*) → bool

Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.9 Annotation Class

The *muspy.Annotation* class is a container for annotations. For flexibility, *annotation* can hold any type of data.

Attributes	Description	Type	Default
time	Start time	int	
annotation	Annotation of any type		

class muspy.**Annotation**(*time: int*, *annotation: Any*, *group: Optional[str] = None*)

A container for annotation.

time

Start time of the annotation, in time steps or seconds.

Type int

annotation

Annotation of any type.

Type any

group

Group name for better organizing the annotations.

Type str, optional

adjust_time(*func: Callable[[int], int]*, *attr: Optional[str] = None*, *recursive: bool = True*) → BaseType

Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.10 Note Class

The *muspy.Note* class is a container for musical notes.

Attributes	Description	Type	Default
time	Start time	int	
duration	Note duration, in time steps	int	
pitch	Note pitch as a MIDI note number	int (0-127)	
velocity	Note velocity	int (0-127)	

Hint: *muspy.Note* has a property *end* with setter and getter implemented, which can be handy sometimes.

class muspy.**Note**(*time: int*, *pitch: int*, *duration: int*, *velocity: Optional[int] = None*, *pitch_str: Optional[str] = None*)

A container for note.

time

Start time of the note, in time steps.

Type int

pitch

Note pitch, as a MIDI note number.

Type int

duration

Duration of the note, in time steps.

Type int

velocity

Note velocity. Defaults to *muspy.DEFAULT_VELOCITY*.

Type int, optional

pitch_str

Note pitch as a string, useful for distinguishing C# and Db.

Type str

adjust_time(*func: Callable[[int], int]*, *attr: Optional[str] = None*, *recursive: bool = True*) → BaseType

Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

clip(*lower: int = 0*, *upper: int = 127*) → muspy.classes.Note

Clip the velocity of the note.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

end
End time of the note.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType
Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

start
Start time of the note.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

transpose(*semitone: int*) → muspy.classes.Note
Transpose the note by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the note. A positive value raises the pitch, while a negative value lowers the pitch.

Returns

Return type Object itself.

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.2.11 Chord Class

The *muspy.Chord* class is a container for chords.

Attributes	Description	Type	Default
time	Start time	int	
duration	Chord duration, in time steps	int	
pitch	Note pitches as MIDI note numbers	list of int (0-127)	[]
velocity	Chord velocity	int (0-127)	

Hint: *muspy.Chord* has a property *end* with setter and getter implemented, which can be handy sometimes.

class muspy.**Chord**(*time: int, pitches: List[int], duration: int, velocity: Optional[int] = None, pitches_str: Optional[List[int]] = None*)
A container for chord.

time
Start time of the chord, in time steps.

Type int

pitches
Note pitches, as MIDI note numbers.

Type list of int

duration
Duration of the chord, in time steps.

Type int

velocity
Chord velocity. Defaults to *muspy.DEFAULT_VELOCITY*.

Type int, optional

pitches_str
Note pitches as strings, useful for distinguishing C# and Db.

Type list of str

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType
Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

clip(*lower: int = 0, upper: int = 127*) → muspy.classes.Chord
Clip the velocity of the chord.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

end
End time of the chord.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType
Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None, recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

start
Start time of the chord.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

transpose(*semitone: int*) → muspy.classes.Chord
Transpose the notes by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

7.3 Timing in MusPy

In MusPy, the *metrical timing* is used. That is, time is stored in musically-meaningful unit (e.g., beats, quarter notes). For playback ability, additional resolution and tempo information is needed.

In a metrical timing system, the smallest unit of time is a factor of a beat, which depends on the time signatures and is set to a quarter note by default. We will refer to this smallest unit of time as a *time step*.

Here is the formula relating the metrical and the absolute timing systems.

$$absolute\_time = \frac{60 \times tempo}{resolution} \times metrical\_time$$

Here, *resolution* is the number of time steps per beat and *tempo* is the current tempo (in quarters per minute, or qpm). These two values are stored in a *muspy.Music* object as attributes `music.resolution` and `music.tempos`.

The following are some illustrations of the relationships between time steps and time.

When reading a MIDI file, `music.resolution` is set to the pulses per quarter note (a.k.a., PPQ, PPQN, ticks per beat). When reading a MusicXML file, `music.resolution` is set to the *division* attribute, which determines the number of divisions per quarter note. When multiple division attributes are found, `music.resolution` is set to the least common multiple of them.

7.4 Input/Output Interfaces

MusPy provides three type of data I/O interfaces.

- Common symbolic music formats: `muspy.read_*` and `muspy.write_*`
- MusPy's native JSON and YAML formats: `muspy.load_*` and `muspy.save_*`
- Other symbolic music libraries: `muspy.from_*` and `muspy.to_*`

7.4.1 MIDI I/O Interface

`muspy.`**`read_midi`**(*path: Union[str, pathlib.Path], backend: str = 'mido', duplicate_note_mode: str = 'fifo'*) → muspy.music.Music

Read a MIDI file into a Music object.

Parameters

- **`path`** (*str or Path*) – Path to the MIDI file to read.
- **`backend`** (*{'mido', 'pretty_midi'}*) – Backend to use.
- **`duplicate_note_mode`** (*{'fifo', 'lifo, 'close_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'. Only used when *backend='mido'*.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close_all': close all note on messages

Returns Converted Music object.

Return type *muspy.Music*

`muspy.`**`write_midi`**(*path: Union[str, pathlib.Path], music: Music, backend: str = 'mido', **kwargs*)

Write a Music object to a MIDI file.

Parameters

- **`path`** (*str or Path*) – Path to write the MIDI file.
- **`music`** (*muspy.Music*) – Music object to write.
- **`backend`** (*{'mido', 'pretty_midi'}*) – Backend to use. Defaults to 'mido'.

7.4.2 MusicXML Interface

muspy.**read_musicxml**(*path: Union[str, pathlib.Path], compressed: Optional[bool] = None*) → muspy.music.Music

Read a MusicXML file into a Music object.

Parameters **path** (*str or Path*) – Path to the MusicXML file to read.

Returns Converted Music object.

Return type *muspy.Music*

Notes

Grace notes and unpitched notes are not supported.

muspy.**write_musicxml**(*path: Union[str, pathlib.Path], music: Music, compressed: Optional[bool] = None*)

Write a Music object to a MusicXML file.

Parameters

- **path** (*str or Path*) – Path to write the MusicXML file.
- **music** (*muspy.Music*) – Music object to write.
- **compressed** (*bool, optional*) – Whether to write to a compressed MusicXML file. If None, infer from the extension of the filename ('.xml' and '.musicxml' for an uncompressed file, '.mxl' for a compressed file).

7.4.3 ABC Interface

muspy.**read_abc**(*path: Union[str, pathlib.Path], number: Optional[int] = None, resolution=24*) → List[muspy.music.Music]

Return an ABC file into Music object(s) using music21 backend.

Parameters

- **path** (*str or Path*) – Path to the ABC file to read.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object(s).

Return type list of *muspy.Music*

muspy.**write_abc**(*path: Union[str, pathlib.Path], music: Music*)

Write a Music object to a ABC file.

Parameters

- **path** (*str or Path*) – Path to write the ABC file.
- **music** (*muspy.Music*) – Music object to write.

7.4.4 JSON Interface

muspy.**load_json**(*path: Union[str, pathlib.Path]*) → muspy.music.Music
Load a JSON file into a Music object.

Parameters **path** (*str or Path*) – Path to the file to load.

Returns Loaded Music object.

Return type *muspy.Music*

muspy.**save_json**(*path: Union[str, pathlib.Path], music: Music*)
Save a Music object to a JSON file.

Parameters

- **path** (*str or Path*) – Path to save the JSON file.
- **music** (*muspy.Music*) – Music object to save.

Note: A JSON schema is available for validating a JSON fiule against MusPy's format.

7.4.5 YAML Interface

muspy.**load_json**(*path: Union[str, pathlib.Path]*) → muspy.music.Music
Load a JSON file into a Music object.

Parameters **path** (*str or Path*) – Path to the file to load.

Returns Loaded Music object.

Return type *muspy.Music*

muspy.**save_json**(*path: Union[str, pathlib.Path], music: Music*)
Save a Music object to a JSON file.

Parameters

- **path** (*str or Path*) – Path to save the JSON file.
- **music** (*muspy.Music*) – Music object to save.

Note: A YAML schema is available for validating a YAML file against MusPy's format.

7.4.6 Mido Interface

muspy.**from_mido**(*midi: mido.midifiles.midifiles.MidiFile, duplicate_note_mode: str = 'fifo'*) → muspy.music.Music
Return a mido MidiFile object as a Music object.

Parameters

- **midi** (mido.MidiFile) – Mido MidiFile object to convert.
- **duplicate_note_mode** (*{'fifo', 'lifo, 'close_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'.

- 'fifo' (first in first out): close the earliest note on
- 'lifo' (first in first out):close the latest note on
- 'close_all': close all note on messages

Returns Converted Music object.

Return type *muspy.Music*

muspy.**to_mido**(*music: Music*, *use_note_on_as_note_off: bool = True*)
Return a Music object as a MidiFile object.

Parameters

- **music** (*muspy.Music* object) – Music object to convert.
- **use_note_on_as_note_off** (*bool*) – Whether to use a note on message with zero velocity instead of a note off message.

Returns Converted MidiFile object.

Return type mido.MidiFile

7.4.7 music21 Interface

muspy.**from_music21**(*stream: music21.stream.Stream*, *resolution=24*) → Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]
Return a music21 Stream object as Music or Track object(s).

Parameters

- **stream** (*music21.stream.Stream*) – Stream object to convert.
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music or Track object(s).

Return type *muspy.Music* or *muspy.Track*

muspy.**to_music21**(*music: Music*) → music21.stream.Score
Return a Music object as a music21 Score object.

Parameters **music** (*muspy.Music*) – Music object to convert.

Returns Converted music21 Score object.

Return type *music21.stream.Score*

7.4.8 pretty_midi Interface

muspy.**from_pretty_midi**(*midi: pretty_midi.pretty_midi.PrettyMIDI*) → muspy.music.Music
Return a pretty_midi PrettyMIDI object as a Music object.

Parameters **midi** (pretty_midi.PrettyMIDI) – PrettyMIDI object to convert.

Returns Converted Music object.

Return type *muspy.Music*

muspy.**to_pretty_midi**(*music: Music*) → pretty_midi.pretty_midi.PrettyMIDI
Return a Music object as a PrettyMIDI object.

Tempo changes are not supported yet.

Parameters **music** (*muspy.Music* object) – Music object to convert.

Returns Converted PrettyMIDI object.

Return type pretty_midi.PrettyMIDI

7.4.9 Pypianoroll Interface

muspy.**from_pypianoroll**(*multitrack: pypianoroll.multitrack.Multitrack*, *default_velocity: int = 64*) → muspy.music.Music
Return a Pypianoroll Multitrack object as a Music object.

Parameters

- **multitrack** (pypianoroll.Multitrack) – Pypianoroll Multitrack object to convert.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns **music** – Converted MusPy Music object.

Return type *muspy.Music*

muspy.**to_pypianoroll**(*music: Music*) → pypianoroll.multitrack.Multitrack
Return a Music object as a Multitrack object.

Parameters **music** (*muspy.Music*) – Music object to convert.

Returns **multitrack** – Converted Multitrack object.

Return type pypianoroll.Multitrack

7.5 Datasets

MusPy provides an easy-to-use dataset management system. Each supported dataset comes with a class inherited from the base MusPy Dataset class. MusPy also provides interfaces to PyTorch and TensorFlow for creating input pipelines for machine learning. Here is an example of preparing training data in the piano-roll representation from the NES Music Database using MusPy.

```
import muspy

# Download and extract the dataset
nes = muspy.NESMusicDatabase("data/nes/", download_and_extract=True)

# Convert the dataset to MusPy Music objects
nes.convert()

# Iterate over the dataset
for music in nes:
    do_something(music)

# Convert to a PyTorch dataset
dataset = nes.to_pytorch_dataset(representation="pianoroll")
```

7.5.1 Iterating over a MusPy Dataset object

Here is an illustration of the two internal processing modes for iterating over a MusPy Dataset object.

7.5.2 Supported Datasets

Here is a list of the supported datasets.

Dataset	Format	Hours	Songs	Genre	Melody	Chords	Multitrack
Lakh MIDI Dataset	MIDI	>5000	174,533	misc	*	*	*
MAESTRO Dataset	MIDI	201.21	1,282	classical			
Wikifonia Lead Sheet Dataset	MusicXML	198.40	6,405	misc	O	O	
Essen Folk Song Dataset	ABC	56.62	9,034	folk	O	O	
NES Music Database	MIDI	46.11	5,278	game	O		O
Hymnal Tune Dataset	MIDI	18.74	1,756	hymn	O		
Hymnal Dataset	MIDI	17.50	1,723	hymn			
music21's Corpus	misc	16.86	613	misc	*		*
Nottingham Database	ABC	10.54	1,036	folk	O	O	
music21's JSBach Corpus	MusicXML	3.46	410	classical			O
JSBach Chorale Dataset	MIDI	3.21	382	classical			O

(Asterisk marks indicate partial support.)

7.5.3 Base Dataset Classes

Here are the two base classes for MusPy datasets.

class muspy.**Dataset**

Base class for MusPy datasets.

To build a custom dataset, it should inherit this class and overide the methods `__getitem__` and `__len__` as well as the class attribute `_info`. `__getitem__` should return the `i`-th data sample as a *muspy.Music* object. `__len__` should return the size of the dataset. `_info` should be a *muspy.DatasetInfo* instance storing the dataset information.

classmethod citation()

Print the citation infomation.

classmethod info()

Return the dataset infomation.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)

Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.

- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

split (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset (*factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]
Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

class muspy.**RemoteDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False*)

Base class for remote MusPy datasets.

This class extends *muspy.Dataset* to support remote datasets. To build a custom remote dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set the class attribute `_sources` to the URLs to the source files (see Notes).

root

Root directory of the dataset.

Type str or Path

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.

Raises RuntimeError: – If `download_and_extract` is False but file `{root}/.muspy.success` does not exist (see below).

Important: `muspy.Dataset.exists()` depends solely on a special file named `.muspy.success` in directory `{root}/_converted/`. This file serves as an indicator for the existence and integrity of the

dataset. It will automatically be created if the dataset is successfully downloaded and extracted by `muspy.Dataset.download_and_extract()`. If the dataset is downloaded manually, make sure to create the `.muspy.success` file in directory `{root}/_converted/` to prevent errors.

Notes

The class attribute `_sources` is a dictionary storing the following information of each source file.

- filename (str): Name to save the file.
- url (str): URL to the file.
- archive (bool): Whether the file is an archive.
- md5 (str, optional): Expected MD5 checksum of the file.

Here is an example.:

```
_sources = {
    "example": {
        "filename": "example.tar.gz",
        "url": "https://www.example.com/example.tar.gz",
        "archive": True,
        "md5": None,
    }
}
```

See also:

muspy.Dataset Base class for MusPy datasets.

classmethod citation()
Print the citation infomation.

download() → RemoteDatasetType
Download the source datasets.

Returns

Return type Object itself.

download_and_extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archives.

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

Notes

Equivalent to `RemoteDataset.download().extract(cleanup)`.

exists() → bool
Return True if the dataset exists, otherwise False.

extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archive(s).

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

classmethod info()
Return the dataset infomation.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

source_exists() → bool
Return True if all the sources exist, otherwise False.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]
Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to numpy.random.RandomState, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to numpy.random.RandomState, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

7.5.4 Local Dataset Classes

Here are the classes for local datasets.

class muspy.**FolderDataset**(*root: Union[str, pathlib.Path], convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Class for datasets storing files in a folder.

This class extends *muspy.Dataset* to support folder datasets. To build a custom folder dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set class attribute `_extension` to the extension to look for when building the dataset and set `read` to a callable that takes as inputs a filename of a source file and return the converted Music object.

root

Root directory of the dataset.

Type str or Path

Parameters

- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

---

Important: *muspy.FolderDataset.converted_exists()* depends solely on a special file named `.muspy.success` in the folder `{root}/_converted/`, which serves as an indicator for the existence and integrity of the converted dataset. If the converted dataset is built by *muspy.FolderDataset.convert()*, the `.muspy.success` file will be created as well. If the converted dataset is created manually, make sure to create the `.muspy.success` file in the folder `{root}/_converted/` to prevent errors.

Notes

Two modes are available for this dataset. When the on-the-fly mode is enabled, a data sample is converted to a music object on the fly when being indexed. When the on-the-fly mode is disabled, a data sample is loaded from the precomputed converted data.

See also:

muspy.Dataset Base class for MusPy datasets.

classmethod citation()

Print the citation infomation.

convert(*kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*) → FolderDatasetType

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Returns

Return type Object itself.

converted_dir
Path to the root directory of the converted dataset.

converted_exists() → bool
Return True if the saved dataset exists, otherwise False.

exists() → bool
Return True if the dataset exists, otherwise False.

classmethod info()
Return the dataset infomation.

load(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Load a file into a Music object.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Any*) → muspy.music.Music
Read a file into a Music object.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

split (*filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*) → Dict[str, List[int]]

Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset (*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset (*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to numpy.random.RandomState, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

use_converted() → FolderDatasetType
Disable on-the-fly mode and use converted data.

Returns

Return type Object itself.

class muspy.**MusicDataset**(*root: Union[str, pathlib.Path], kind: str = 'json'*)
Class for datasets of MusPy JSON/YAML files.

root
Root directory of the dataset.

Type str or Path

kind
File format of the data. Defaults to ‘json’.

Type {‘json’, ‘yaml’}, optional

See also:

muspy.Dataset Base class for MusPy datasets.

classmethod citation()
Print the citation infomation.

classmethod info()
Return the dataset infomation.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to ‘json’.

- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

split (*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset (*factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]
Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

class muspy.**ABCFolderDataset**(*root: Union[str, pathlib.Path]*, *convert: bool = False*, *kind: str = 'json'*, *n_jobs: int = 1*, *ignore_exceptions: bool = True*, *use_converted: Optional[bool] = None*)

Class for datasets storing ABC files in a folder.

See also:

muspy.FolderDataset Class for datasets storing files in a folder.

classmethod citation()

Print the citation infomation.

convert(*kind: str = 'json'*, *n_jobs: int = 1*, *ignore_exceptions: bool = True*) → FolderDatasetType

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to ‘json’.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Returns

Return type Object itself.

converted_dir
Path to the root directory of the converted dataset.

converted_exists() → bool
Return True if the saved dataset exists, otherwise False.

exists() → bool
Return True if the dataset exists, otherwise False.

classmethod info()
Return the dataset infomation.

load(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Load a file into a Music object.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Tuple[str, Tuple[int, int]]*) → muspy.music.Music
Read a file into a Music object.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and

test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

use_converted() → FolderDatasetType
Disable on-the-fly mode and use converted data.

Returns

Return type Object itself.

7.5.5 Remote Dataset Classes

Here are the classes for remote datasets.

class muspy.**RemoteFolderDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Base class for remote datasets stoting files in a folder.

root
Root directory of the dataset.

Type str or Path

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.
- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to ‘json’.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

See also:

muspy.FolderDataset Class for datasets storing files in a folder.

muspy.RemoteDataset Base class for remote MusPy datasets.

classmethod citation()
Print the citation infomation.

convert(*kind: str = 'json'*, *n_jobs: int = 1*, *ignore_exceptions: bool = True*) → FolderDatasetType
Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Returns

Return type Object itself.

converted_dir
Path to the root directory of the converted dataset.

converted_exists() → bool
Return True if the saved dataset exists, otherwise False.

download() → RemoteDatasetType
Download the source datasets.

Returns

Return type Object itself.

download_and_extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archives.

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

Notes

Equivalent to `RemoteDataset.download().extract(cleanup)`.

exists() → bool
Return True if the dataset exists, otherwise False.

extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archive(s).

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

classmethod info()
Return the dataset infomation.

load(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Load a file into a Music object.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: str*) → muspy.music.Music
Read a file into a Music object.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

source_exists() → bool
Return True if all the sources exist, otherwise False.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

use_converted() → FolderDatasetType
Disable on-the-fly mode and use converted data.

Returns

Return type Object itself.

class muspy.**RemoteMusicDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, kind: str = 'json'*)
Base class for remote datasets of MusPy JSON/YAML files.

root
Root directory of the dataset.

Type str or Path

kind
File format of the data. Defaults to ‘json’.

Type {‘json’, ‘yaml’}, optional

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.

See also:

muspy.MusicDataset Class for datasets of MusPy JSON/YAML files.

muspy.RemoteDataset Base class for remote MusPy datasets.

classmethod citation()
Print the citation infomation.

download() → RemoteDatasetType
Download the source datasets.

Returns

Return type Object itself.

download_and_extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archives.

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

Notes

Equivalent to `RemoteDataset.download().extract(cleanup)`.

exists() → bool
Return True if the dataset exists, otherwise False.

extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archive(s).

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

classmethod info()
Return the dataset infomation.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

source_exists() → bool
Return True if all the sources exist, otherwise False.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

class muspy.**RemoteABCFolderDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Base class for remote datasets storing ABC files in a folder.

See also:

muspy.ABCFolderDataset Class for datasets storing ABC files in a folder.

muspy.RemoteDataset Base class for remote MusPy datasets.

classmethod citation()
Print the citation infomation.

convert(*kind: str = 'json'*, *n_jobs: int = 1*, *ignore_exceptions: bool = True*) → FolderDatasetType
Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to ‘json’.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Returns

Return type Object itself.

converted_dir
Path to the root directory of the converted dataset.

converted_exists() → bool
Return True if the saved dataset exists, otherwise False.

download() → RemoteDatasetType
Download the source datasets.

Returns

Return type Object itself.

download_and_extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archives.

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

Notes

Equivalent to `RemoteDataset.download().extract(cleanup)`.

exists() → bool
Return True if the dataset exists, otherwise False.

extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archive(s).

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

classmethod info()
Return the dataset infomation.

load(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Load a file into a Music object.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Tuple[str, Tuple[int, int]]*) → muspy.music.Music
Read a file into a Music object.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

source_exists() → bool
Return True if all the sources exist, otherwise False.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and

test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

use_converted() → FolderDatasetType
Disable on-the-fly mode and use converted data.

Returns

Return type Object itself.

7.6 Representations

MusPy supports several common representations for symbolic music. Here is a comparison of them.

Representation	Shape	Values	Default configurations
Pitch-based	T x 1	{0, 1, . . . , 129}	128 note-ons, 1 hold, 1 rest (support only monophonic music)
Piano-roll	T x 128	{0, 1} or N	{0,1} for binary piano rolls; N for piano rolls with velocities
Event-based	M x 1	{0, 1, . . . , 387}	128 note-ons, 128 note-offs, 100 tick shifts, 32 velocities
Note-based	N x 4	N	List of (*time*, *pitch*, *duration*, *velocity*) tuples

Note that T, M, and N denote the numbers of time steps, events and notes, respectively.

MusPy's representation module supports two types of two APIs—Functional API and Processor API. Take the pitch-based representation for example.

- The Functional API provide two functions: - *muspy.to_pitch_representation()*: Convert a Music object into pitch-based representation - *muspy.from_pitch_representation()*: Return a Music object converted from pitch-based representation
- The Processor API provides the class *muspy.PitchRepresentationProcessor*, which provides two methods: - *muspy.PitchRepresentationProcessor.encode()*: Convert a Music object into pitch-based representation - *muspy.PitchRepresentationProcessor.decode()*: Return a Music object converted from pitch-based representation

7.6.1 Pitch-based Representation

muspy.**to_pitch_representation**(*music: Music*, *use_hold_state: bool = False*) → numpy.ndarray
Encode a Music object into pitch-based representation.

The pitch-based represetantion represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_hold_state** (*bool*) – Whether to use a special state for holds. Defaults to False.

Returns Encoded array in pitch-based representation.

Return type ndarray, dtype=uint8, shape=(?, 1)

muspy.**from_pitch_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_hold_state: bool = False*, *default_velocity: int = 64*) → muspy.music.Music

Decode pitch-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in pitch-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_hold_state** (*bool*) – Whether to use a special state for holds. Defaults to False.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

class muspy.**PitchRepresentationProcessor**(*use_hold_state: bool = False*, *default_velocity: int = 64*)

Pitch-based representation processor.

The pitch-based represetantion represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

use_hold_state
Whether to use a special state for holds. Defaults to False.

Type bool

default_velocity
Default velocity value to use when decoding. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode pitch-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in pitch-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_pitch_representation() Return a Music object converted from pitch-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into pitch-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in pitch-based representation.

Return type ndarray (np.uint8)

See also:

muspy.to_pitch_representation() Convert a Music object into pitch-based representation.

7.6.2 Piano-roll Representation

muspy.**to_pianoroll_representation**(*music: Music*, *encode_velocity: bool = True*) → numpy.ndarray
Encode notes into piano-roll representation.

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **encode_velocity** (*bool*) – Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

Returns Encoded array in piano-roll representation.

Return type ndarray, dtype=uint8 or bool, shape=(?, 128)

muspy.**from_pianoroll_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *encode_velocity: bool = True*, *default_velocity: int = 64*) → muspy.music.Music
Decode pitch-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in piano-roll representation to decode. Will be casted to integer if not of integer type. If *encode_velocity* is True, will be casted to boolean if not of boolean type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode velocities. Defaults to True.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

class muspy.**PianoRollRepresentationProcessor**(*encode_velocity: bool = True*, *default_velocity: int = 64*)

Piano-roll representation processor.

The piano-roll represetantion represents music as a time-pitch matrix, where the columns are the time steps and the rows are the pitches. The values indicate the presence of pitches at different time steps. The output shape is T x 128, where T is the number of time steps.

encode_velocity
Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

Type bool

default_velocity
Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode piano-roll representation into a Music object.

Parameters **array** (*ndarray*) – Array in piano-roll representation to decode. Cast to integer if not of integer type. If *encode_velocity* is True, casted to boolean if not of boolean type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_pianoroll_representation() Return a Music object converted from piano-roll representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into piano-roll representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in piano-roll representation.

Return type ndarray (np.uint8)

See also:

muspy.to_pianoroll_representation() Convert a Music object into piano-roll representation.

7.6.3 Event-based Representation

muspy.**to_event_representation**(*music: Music*, *use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *encode_velocity: bool = False*, *force_velocity_event: bool = True*, *max_time_shift: int = 100*, *velocity_bins: int = 32*) → numpy.ndarray

Encode a Music object into event-based representation.

The event-based represetantion represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is M x 1, where M is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-one events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_single_note_off_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use_end_of_sequence_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode velocities.
- **force_velocity_event** (*bool*) – Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.
- **max_time_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity_bins** (*int*) – Number of velocity bins to use. Defaults to 32.

Returns Encoded array in event-based representation.

Return type ndarray, dtype=uint16, shape=(?, 1)

muspy.**from_event_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *max_time_shift: int = 100*, *velocity_bins: int = 32*, *default_velocity: int = 64*) → muspy.music.Music

Decode event-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in event-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.

- **use_single_note_off_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, a note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use_end_of_sequence_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **max_time_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity_bins** (*int*) – Number of velocity bins to use. Defaults to 32.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

class muspy.**EventRepresentationProcessor**(*use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *encode_velocity: bool = False*, *force_velocity_event: bool = True*, *max_time_shift: int = 100*, *velocity_bins: int = 32*, *default_velocity: int = 64*)

Event-based representation processor.

The event-based represetantion represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is M x 1, where M is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-one events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

use_single_note_off_event
Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.

Type bool

use_end_of_sequence_event
Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.

Type bool

encode_velocity
Whether to encode velocities.

Type bool

force_velocity_event
Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

Type bool

max_time_shift
Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.

Type int

velocity_bins
Number of velocity bins to use. Defaults to 32.

Type int

default_velocity
Default velocity value to use when decoding. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode event-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_event_representation() Return a Music object converted from event-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into event-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in event-based representation.

Return type ndarray (np.uint16)

See also:

muspy.to_event_representation() Convert a Music object into event-based representation.

7.6.4 Note-based Representation

muspy.**to_note_representation**(*music: Music*, *use_start_end: bool = False*, *encode_velocity: bool = True*, *dtype: Union[numpy.dtype, type, str] = <class 'int'>*) → numpy.ndarray
Encode a Music object into note-based representation.

The note-based represetantion represents music as a sequence of (time, pitch, duration, velocity) tuples. For example, a note Note(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 60, 4, 64). The output shape is N * D, where N is the number of notes and D is 4 when *encode_velocity* is True, otherwise D is 3. The values of the second dimension represent time, pitch, duration and velocity (discarded when *encode_velocity* is False).

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_start_end** (*bool*) – Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **dtype** (*dtype, type or str*) – Data type of the return array. Defaults to int.

Returns Encoded array in note-based representation.

Return type ndarray, shape=(?, 3 or 4)

muspy.**from_note_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_start_end: bool = False*, *encode_velocity: bool = True*, *default_velocity: int = 64*) → muspy.music.Music

Decode note-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in note-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_start_end** (*bool*) – Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **default_velocity** (*int*) – Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

class muspy.**NoteRepresentationProcessor**(*use_start_end: bool = False*, *encode_velocity: bool = True*, *dtype: Union[numpy.dtype, type, str] = <class 'int'>*, *default_velocity: int = 64*)

Note-based representation processor.

The note-based represetantion represents music as a sequence of (pitch, time, duration, velocity) tuples. For example, a note Note(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 4, 60, 64). The output shape is L * D, where L is th number of notes and D is 4 when *encode_velocity* is True, otherwise D is 3. The values of the second dimension represent pitch, time, duration and velocity (discarded when *encode_velocity* is False).

use_start_end
Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.

Type bool

encode_velocity
Whether to encode note velocities. Defaults to True.

Type bool

dtype
Data type of the return array. Defaults to int.

Type dtype, type or str

default_velocity
Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode note-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in note-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_note_representation() Return a Music object converted from note-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into note-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in note-based representation.

Return type ndarray (np.uint8)

See also:

muspy.to_note_representation() Convert a Music object into note-based representation.

7.7 Synthesis

muspy.**write_audio**(*path: Union[str, pathlib.Path], music: Music, soundfont_path: Union[str, pathlib.Path, None] = None, rate: int = 44100, audio_format: Optional[str] = None*)
Write a Music object to an audio file.

Supported formats include WAV, AIFF, FLAC and OGA.

Parameters

- **path** (*str or Path*) – Path to write the audio file.
- **music** (*muspy.Music*) – Music object to write.
- **soundfont_path** (*str or Path, optional*) – Path to the soundfount file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.
- **audio_format** (*str, {'wav', 'aiff', 'flac', 'oga'}, optional*) – File format to write. If None, infer it from the extension.

muspy.**synthesize**(*music: Music, soundfont_path: Union[str, pathlib.Path, None] = None, rate: int = 44100*) → numpy.ndarray
Synthesize a Music object to raw audio.

Parameters

- **music** (*muspy.Music*) – Music object to write.

- **soundfont_path** (*str or Path, optional*) – Path to the soundfount file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.

Returns Synthesized waveform.

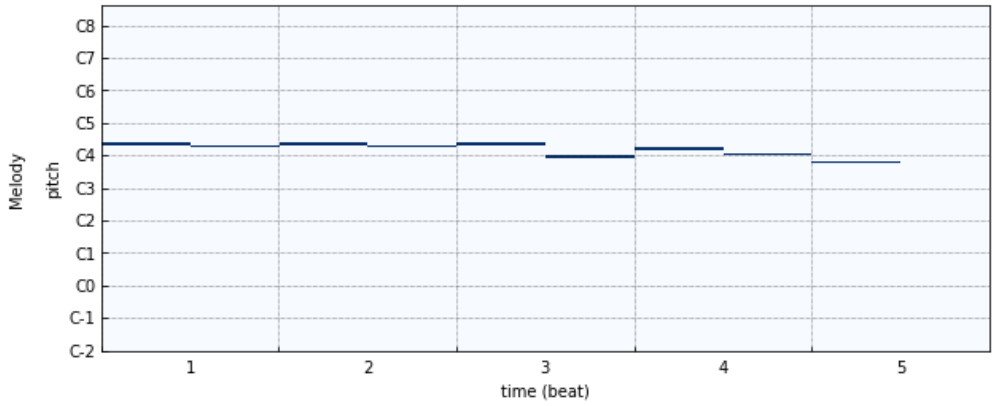
Return type ndarray, dtype=int16, shape=(?, 2)

7.8 Visualization

MusPy supports two visualization tools. Both use Matplotlib as the backend for flexibility.

7.8.1 Piano-roll Visualization

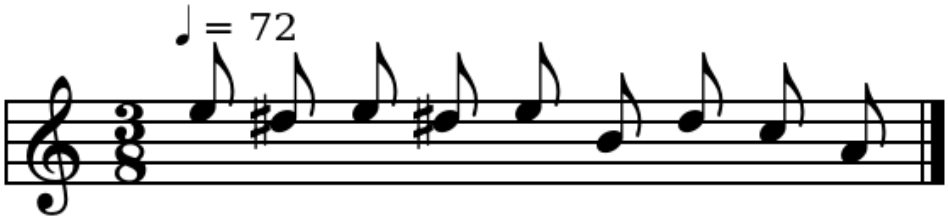
The piano-roll visualization is made possible with the Pypianoroll library.



muspy.**show_pianoroll**(*music: Music*, ***kwargs*)
Show pianoroll visualization.

7.8.2 Score Visualization

The score visualization is made possible with the Bravura font.



muspy.**show_score**(*music: Music*, *figsize: Optional[Tuple[float, float]] = None*, *clef: str = 'treble'*, *clef_octave: Optional[int] = 0*, *note_spacing: Optional[int] = None*, *font_path: Union[str, pathlib.Path, None] = None*, *font_scale: Optional[float] = None*) → muspy.visualization.score.ScorePlotter
Show score visualization.

Parameters

- **music** (*muspy.Music*) – Music object to show.
- **figsize** (*(float, float), optional*) – Width and height in inches. Defaults to Matplotlib configuration.
- **clef** (*str, {'treble', 'alto', 'bass'}*) – Clef type. Defaults to a treble clef.
- **clef_octave** (*int*) – Clef octave. Defaults to zero.
- **note_spacing** (*int, optional*) – Spacing of notes. Defaults to 4.
- **font_path** (*str or Path, optional*) – Path to the music font. Defaults to the path to the built-in Bravura font.
- **font_scale** (*float, optional*) – Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

Returns A ScorePlotter object that handles the score.

Return type *muspy.ScorePlotter*

muspy.**ScorePlotter**(*fig: matplotlib.figure.Figure*, *ax: matplotlib.axes._axes.Axes*, *resolution: int*, *note_spacing: Optional[int] = None*, *font_path: Union[str, pathlib.Path, None] = None*, *font_scale: Optional[float] = None*)

A plotter that handles the score visualization.

muspy.**fig**
Figure object to plot the score on.

Type matplotlib.figure.Figure

muspy.**axes**
Axes object to plot the score on.

Type matplotlib.axes.Axes

muspy.**resolution**
Time steps per quarter note.

Type int

muspy.**note_spacing**
Spacing of notes. Defaults to 4.

Type int, optional

muspy.**font_path**
Path to the music font. Defaults to the path to the downloaded Bravura font.

Type str or Path, optional

muspy.**font_scale**
Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

Type float, optional

7.9 Metrics

MusPy provides several objective metrics proposed in the literature, summarized as follows.

- **Pitch-related metrics**: pitch_range, n_pitches_used, n_pitch_classes_used, polyphony, polyphony rate, pitch-in-scale rate, scale consistency, pitch entropy and pitch class entropy.

- **Rhythm-related metrics**: empty-beat rate, drum-in-pattern rate, drum pattern consistency and groove consistency.
- **Other metrics**: empty_measure_rate.

These objective metrics could be used to evaluate a music generation system by comparing the statistical difference between the training data and the generated samples.

7.9.1 Pitch-related metrics

muspy.**pitch_range**(*music: muspy.music.Music*) → int
Return the pitch range.

Drum tracks are ignored. Return zero if no note is found.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch range.

Return type int

muspy.**n_pitches_used**(*music: muspy.music.Music*) → int
Return the number of unique pitches used.

Drum tracks are ignored.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Number of unique pitch used.

Return type int

See also:

muspy.n_pitch_class_used() Compute the number of unique pitch classes used.

muspy.**n_pitch_classes_used**(*music: muspy.music.Music*) → int
Return the number of unique pitch classes used.

Drum tracks are ignored.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Number of unique pitch classes used.

Return type int

See also:

muspy.n_pitches_used() Compute the number of unique pitches used.

muspy.**polyphony**(*music: muspy.music.Music*) → float
Return the average number of pitches being played concurrently.

The polyphony is defined as the average number of pitches being played at the same time, evaluated only at time steps where at least one pitch is on. Drum tracks are ignored. Return NaN if no note is found.

$$polyphony = \frac{\#(pitches\_when\_at\_least\_one\_pitch\_is\_on)}{\#(time\_steps\_where\_at\_least\_one\_pitch\_is\_on)}$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Polyphony.

Return type float

See also:

muspy.polyphony_rate() Compute the ratio of time steps where multiple pitches are on.

muspy.**polyphony_rate**(*music: muspy.music.Music*, *threshold: int = 2*) → float

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$polyphony\_rate = \frac{\#(time\_steps\_where\_multiple\_pitches\_are\_on)}{\#(time\_steps)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

Returns Polyphony rate.

Return type float

See also:

muspy.polyphony() Compute the average number of pitches being played at the same time.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**pitch_in_scale_rate**(*music: muspy.music.Music*, *root: int*, *mode: str*) → float

Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_in\_scale\_rate = \frac{\#(notes\_in\_scale)}{\#(notes)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str, {'major', 'minor'}*) – Mode of the scale.

Returns Pitch-in-scale rate.

Return type float

See also:

muspy.scale_consistency() Compute the largest pitch-in-class rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**scale_consistency**(*music: muspy.music.Music*) → float

Return the largest pitch-in-scale rate.

The scale consistency is defined as the largest pitch-in-scale rate over all major and minor scales. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$scale\_consistency = \max_{root,mode} pitch\_in\_scale\_rate(root, mode)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Scale consistency.

Return type float

See also:

muspy.pitch_in_scale_rate() Compute the ratio of pitches in a certain musical scale.

References

1. Olof Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training," in NeuIPS Workshop on Constructive Machine Learning, 2016.

muspy.**pitch_entropy**(*music: muspy.music.Music*) → float

Return the entropy of the normalized note pitch histogram.

The pitch entropy is defined as the Shannon entropy of the normalized note pitch histogram. Drum tracks are ignored. Return NaN if no note is found.

$$pitch\_entropy = -\sum_{i=0}^{127} P(pitch = i) \log_2 P(pitch = i)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch entropy.

Return type float

See also:

muspy.pitch_class_entropy() Compute the entropy of the normalized pitch class histogram.

muspy.**pitch_class_entropy**(*music: muspy.music.Music*) → float

Return the entropy of the normalized note pitch class histogram.

The pitch class entropy is defined as the Shannon entropy of the normalized note pitch class histogram. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_class\_entropy = -\sum_{i=0}^{11} P(pitch\_class = i) \times \log_2 P(pitch\_class = i)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch class entropy.

Return type float

See also:

muspy.pitch_entropy() Compute the entropy of the normalized pitch histogram.

References

1. Shih-Lun Wu and Yi-Hsuan Yang, "The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures", in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

7.9.2 Rhythm-related metrics

muspy.**empty_beat_rate**(*music: muspy.music.Music*) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero. This metric is also implemented in Pypianoroll [1].

$$empty\_beat\_rate = \frac{\#(empty\_beats)}{\#(beats)}$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Empty-beat rate.

Return type float

See also:

muspy.empty_measure_rate() Compute the ratio of empty measures.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, "Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls," in Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2018.

muspy.**drum_in_pattern_rate**(*music: muspy.music.Music*, *meter: str*) → float

Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$drum\_in\_pattern\_rate = \frac{\#(drum\_notes\_in\_pattern)}{\#(drum\_notes)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **meter** (*str, {'duple', 'triple'}*) – Meter of the drum pattern.

Returns Drum-in-pattern rate.

Return type float

See also:

muspy.drum_pattern_consistency() Compute the largest drum-in-pattern rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**drum_pattern_consistency**(*music: muspy.music.Music*) → float
Return the largest drum-in-pattern rate.

The drum pattern consistency is defined as the largest drum-in-pattern rate over duple and triple meters. Only drum tracks are considered. Return NaN if no drum note is found.

$$drum\_pattern\_consistency = \max_{meter} drum\_in\_pattern\_rate(meter)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Drum pattern consistency.

Return type float

See also:

muspy.drum_in_pattern_rate() Compute the ratio of drum notes in a certain drum pattern.

muspy.**groove_consistency**(*music: muspy.music.Music*, *measure_resolution: int*) → float
Return the groove consistency.

The groove consistency is defined as the mean hamming distance of the neighboring measures.

$$groove\_consistency = 1 - \frac{1}{T-1}\sum_{i=1}^{T-1} d(G_i, G_{i+1})$$

Here, T is the number of measures, G_i is the binary onset vector of the i-th measure (a one at position that has an onset, otherwise a zero), and $d(G, G')$ is the hamming distance between two vectors G and G'. Note that this metric only works for songs with a constant time signature. Return NaN if the number of measures is less than two. This metric is used in [1].

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure_resolution** (*int*) – Time steps per measure.

Returns Groove consistency.

Return type float

References

1. Shih-Lun Wu and Yi-Hsuan Yang, "The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures", in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

7.9.3 Other metrics

muspy.**empty_measure_rate**(*music: muspy.music.Music*, *measure_resolution: int*) → float

Return the ratio of empty measures.

The empty-measure rate is defined as the ratio of the number of empty measures (where no note is played) to the total number of measures. Note that this metric only works for songs with a constant time signature. Return NaN if song length is zero. This metric is used in [1].

$$empty\_measure\_rate = \frac{\#(empty\_measures)}{\#(measures)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure_resolution** (*int*) – Time steps per measure.

Returns Empty-measure rate.

Return type float

See also:

muspy.empty_beat_rate() Compute the ratio of empty beats.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

7.10 Technical Documentation

These are the detailed technical documentation.

7.10.1 muspy

A toolkit for symbolic music generation.

MusPy is an open source Python library for symbolic music generation. It provides essential tools for developing a music generation system, including dataset management, data I/O, data preprocessing and model evaluation.

Features

- Dataset management system for commonly used datasets with interfaces to PyTorch and TensorFlow.
- Data I/O for common symbolic music formats (e.g., MIDI, MusicXML and ABC) and interfaces to other symbolic music libraries (e.g., music21, mido, pretty_midi and Pypianoroll).
- Implementations of common music representations for music generation, including the pitch-based, the event-based, the piano-roll and the note-based representations.
- Model evaluation tools for music generation systems, including audio rendering, score and piano-roll visualizations and objective metrics.

class muspy.**Base**(***kwargs*)

Base class for MusPy classes.

This is the base class for MusPy classes. It provides two handy I/O methods—*from_dict* and *to_ordered_dict*. It also provides intuitive *__repr__* as well as methods *pretty_str* and *print* for beautifully printing the content.

Hint: To implement a new class in MusPy, please inherit from this class and set the following class variables properly.

- *_attributes*: An OrderedDict with attribute names as keys and their types as values.
- *_optional_attributes*: A list of optional attribute names.
- *_list_attributes*: A list of attributes that are lists.

Take *muspy.Note* for example.:

```
_attributes = OrderedDict(
    [
        ("time", int),
        ("duration", int),
        ("pitch", int),
        ("velocity", int),
        ("pitch_str", str),
    ]
)
_optional_attributes = ["pitch_str"]
```

See also:

muspy.ComplexBase Base class that supports advanced operations on list attributes.

adjust_time(*func: Callable[[int], int], attr: Optional[str] = None, recursive: bool = True*) → BaseType

Adjust the timing of time-stamped objects.

Parameters

- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.
- **attr** (*str*) – Attribute to adjust. Defaults to adjust all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

classmethod from_dict(*dict_: Mapping[KT, VT_co]*) → BaseType

Return an instance constructed from a dictionary.

Instantiate an object whose attributes and the corresponding values are given as a dictionary.

Parameters **dict** (*dict or mapping*) – A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Returns

Return type Constructed object.

is_valid(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute has a valid type and value.

This will recursively apply to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns Whether the attribute has a valid type and value.

Return type bool

See also:

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

is_valid_type(*attr: Optional[str] = None*, *recursive: bool = True*) → bool
Return True if an attribute is of a valid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

- *bool* – Whether the attribute is of a valid type.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

See also:

muspy.Base.validate_type() Raise an error if a certain attribute is of an invalid type.

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

pretty_str(*skip_none: bool = True*) → str
Return the attributes as a string in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns Stored data as a string in a YAML-like format.

Return type str

See also:

muspy.Base.print() Print the attributes in a YAML-like format.

print(*skip_none: bool = True*)
Print the attributes in a YAML-like format.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

See also:

muspy.Base.pretty_str() Return the the attributes as a string in a YAML-like format.

to_ordered_dict(*skip_none: bool = True*) → collections.OrderedDict
Return the object as an OrderedDict.

Return an ordered dictionary that stores the attributes and their values as key-value pairs.

Parameters **skip_none** (*bool*) – Whether to skip attributes with value None or those that are empty lists. Defaults to True.

Returns A dictionary that stores the attributes and their values as key-value pairs, e.g., *{"attr1": value1, "attr2": value2}*.

Return type OrderedDict

validate(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute has an invalid type or value.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid() Return True if an attribute has a valid type and value.

muspy.Base.validate_type() Raise an error if an attribute is of an invalid type.

validate_type(*attr: Optional[str] = None*, *recursive: bool = True*) → BaseType
Raise an error if an attribute is of an invalid type.

This will apply recursively to an attribute's attributes.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

See also:

muspy.Base.is_valid_type() Return True if an attribute is of a valid type.

muspy.Base.validate() Raise an error if an attribute has an invalid type or value.

class muspy.**ComplexBase**(***kwargs*)
Base class that supports advanced operations on list attributes.

This class extend the Base class with advanced operations on list attributes, including *append*, *remove_invalid*, *remove_duplicate* and *sort*.

See also:

muspy.Base Base class for MusPy classes.

append(*obj*) → ComplexBaseType
Append an object to the correseponding list.

This will automatically determine the list attributes to append based on the type of the object.

Parameters **obj** – Object to append.

remove_duplicate(*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Remove duplicate items from a list attribute.

Parameters

- **attr** (*str*) – Attribute to check. Defaults to check all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

remove_invalid(*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Remove invalid items from a list attribute.

Parameters

- **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

sort(*attr: Optional[str] = None*, *recursive: bool = True*) → ComplexBaseType
Sort a list attribute.

Parameters

- **attr** (*str*) – Attribute to sort. Defaults to sort all attributes.
- **recursive** (*bool*) – Whether to apply recursively. Defaults to True.

Returns

Return type Object itself.

class muspy.**Annotation**(*time: int*, *annotation: Any*, *group: Optional[str] = None*)
A container for annotation.

time
Start time of the annotation, in time steps or seconds.

Type int

annotation
Annotation of any type.

Type any

group
Group name for better organizing the annotations.

Type str, optional

class muspy.**Chord**(*time: int, pitches: List[int], duration: int, velocity: Optional[int] = None, pitches_str: Optional[List[int]] = None*)

A container for chord.

time
Start time of the chord, in time steps.

Type int

pitches
Note pitches, as MIDI note numbers.

Type list of int

duration
Duration of the chord, in time steps.

Type int

velocity
Chord velocity. Defaults to *muspy.DEFAULT_VELOCITY*.

Type int, optional

pitches_str
Note pitches as strings, useful for distinguishing C# and Db.

Type list of str

clip(*lower: int = 0*, *upper: int = 127*) → muspy.classes.Chord
Clip the velocity of the chord.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

end
End time of the chord.

start
Start time of the chord.

transpose(*semitone: int*) → muspy.classes.Chord
Transpose the notes by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

class muspy.**KeySignature**(*time: int*, *root: Optional[int] = None*, *mode: Optional[str] = None*, *fifths: Optional[int] = None*, *root_str: Optional[str] = None*)

A container for key signature.

time
Start time of the key signature, in time steps or seconds.

Type int

root
Root of the key signature.

Type int, optional

mode
Mode of the key signature.

Type str, optional

fifths
Number of flats or sharps. Positive numbers for sharps and negative numbers for flats.

Type int, optional

root_str
Root of the key signature as a string.

Type str, optional

class muspy.**Lyric**(*time: int*, *lyric: str*)
A container for lyric.

time
Start time of the lyric, in time steps or seconds.

Type int

lyric
Lyric (sentence, word, syllable, etc.).

Type str

class muspy.**Metadata**(*schema_version: str = '0.0'*, *title: Optional[str] = None*, *creators: Optional[List[str]] = None*, *copyright: Optional[str] = None*, *collection: Optional[str] = None*, *source_filename: Optional[str] = None*, *source_format: Optional[str] = None*)
A container for metadata.

schema_version
Schema version. Defaults to the latest version.

Type str

title
Song title.

Type str, optional

creators
Creator(s) of the song.

Type list of str, optional

copyright
Copyright notice.

Type str, optional

collection
Name of the collection.

Type str, optional

source_filename
Name of the source file.

Type str, optional

source_format
Format of the source file.

Type str, optional

class muspy.**Note**(*time: int*, *pitch: int*, *duration: int*, *velocity: Optional[int] = None*, *pitch_str: Optional[str] = None*)
A container for note.

time
Start time of the note, in time steps.

Type int

pitch
Note pitch, as a MIDI note number.

Type int

duration
Duration of the note, in time steps.

Type int

velocity
Note velocity. Defaults to *muspy.DEFAULT_VELOCITY*.

Type int, optional

pitch_str
Note pitch as a string, useful for distinguishing C# and Db.

Type str

clip(*lower: int = 0*, *upper: int = 127*) → muspy.classes.Note
Clip the velocity of the note.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

end
End time of the note.

start
Start time of the note.

transpose(*semitone: int*) → muspy.classes.Note
Transpose the note by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the note. A positive value raises the pitch, while a negative value lowers the pitch.

Returns

Return type Object itself.

class muspy.**Tempo**(*time: int*, *qpm: float*)
A container for key signature.

time
Start time of the tempo, in time steps.

Type int

qpm
Tempo in qpm (quarters per minute).

Type float

class muspy.**TimeSignature**(*time: int*, *numerator: int*, *denominator: int*)
A container for time signature.

time
Start time of the time signature, in time steps or seconds.

Type int

numerator
Numerator of the time signature.

Type int

denominator
Denominator of the time signature.

Type int

class muspy.**Track**(*program: int = 0*, *is_drum: bool = False*, *name: Optional[str] = None*, *notes: Optional[List[muspy.classes.Note]] = None*, *chords: Optional[List[muspy.classes.Chord]] = None*, *lyrics: Optional[List[muspy.classes.Lyric]] = None*, *annotations: Optional[List[muspy.classes.Annotation]] = None*)
A container for music track.

program
Program number according to General MIDI specification[1]. Defaults to 0 (Acoustic Grand Piano).

Type int, 0-127, optional

is_drum
Whether it is a percussion track. Defaults to False.

Type bool, optional

name
Track name.

Type str, optional

notes
Musical notes. Defaults to an empty list.

Type list of *muspy.Note*, optional

chords
Chords. Defaults to an empty list.

Type list of *muspy.Chord*, optional

annotations
Annotations. Defaults to an empty list.

Type list of *muspy.Annotation*, optional

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

lyrics
Lyrics. Defaults to an empty list.

Type list of *muspy.Lyric*, optional

Note: Indexing a Track object returns the note at a certain index. That is, `track[idx]` returns `track.notes[idx]`. Length of a Track object is the number of notes. That is, `len(track)` returns `len(track.notes)`.

References

clip(*lower: int = 0*, *upper: int = 127*) → muspy.classes.Track
Clip the velocity of each note.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

get_end_time(*is_sorted: bool = False*) → int
Return the time of the last event.

This includes notes, chords, lyrics and annotations.

Parameters **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

transpose(*semitone: int*) → muspy.classes.Track
Transpose the notes by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

muspy.**adjust_resolution**(*music: muspy.music.Music*, *target: Optional[int] = None*, *factor: Optional[float] = None*, *rounding: Union[str, Callable, None] = 'round'*) → muspy.music.Music
Adjust resolution and timing of all time-stamped objects.

Parameters

- **music** (*muspy.Music*) – Object to adjust the resolution.
- **target** (*int, optional*) – Target resolution.
- **factor** (*int or float, optional*) – Factor used to adjust the resolution based on the formula: *new_resolution* = *old_resolution* * *factor*. For example, a factor of 2 double the resolution, and a factor of 0.5 halve the resolution.
- **rounding** (*{'round', 'ceil', 'floor'} or callable, optional*) – Rounding mode. Defaults to 'round'.

muspy.**adjust_time**(*obj: muspy.base.Base*, *func: Callable[[int], int]*) → muspy.base.Base
Adjust the timing of time-stamped objects.

Parameters

- **obj** (*muspy.Music* or *muspy.Track*) – Object to adjust the timing.
- **func** (*callable*) – The function used to compute the new timing from the old timing, i.e., *new_time = func(old_time)*.

See also:

muspy.adjust_resolution() Adjust the resolution and the timing of time-stamped objects.

Note: The resolution are left unchanged.

---

muspy.**append**(*obj1: muspy.base.ComplexBase*, *obj2*) → muspy.base.ComplexBase
Append an object to the correseponding list.

Parameters

- **obj1** (*muspy.Music*, *muspy.Track* or *muspy.Tempo*) – Object to which *obj2* to append.
- **obj2** – Object to be appended to *obj1*.

Notes

- If *obj1* is of type *muspy.Music*, *obj2* can be *muspy.KeySignature*, *muspy.TimeSignature*, *muspy.Lyric*, *muspy.Annotation* or *muspy.Track*.
- If *obj1* is of type *muspy.Track*, *obj2* can be *muspy.Note*, *muspy.Lyric* or *muspy.Annotation*.
- If *obj1* is of type muspy.Timing, *obj2* can be *muspy.Tempo*.

muspy.**clip**(*obj: Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note]*, *lower: int = 0*, *upper: int = 127*) → Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note]
Clip the velocity of each note.

Parameters

- **obj** (*muspy.Music*, *muspy.Track* or *muspy.Note*) – Object to clip.
- **lower** (*int or float, optional*) – Lower bound. Defaults to 0.
- **upper** (*int or float, optional*) – Upper bound. Defaults to 127.

muspy.**get_end_time**(*obj: Union[muspy.music.Music, muspy.classes.Track]*, *is_sorted: bool = False*) → int
Return the the time of the last event in all tracks.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations.

Parameters

- **obj** (*muspy.Music* or *muspy.Track*) – Object to inspect.
- **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

muspy.**get_real_end_time**(*music: muspy.music.Music*, *is_sorted: bool = False*) → float
Return the end time in realtime.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations. Assume 120 qpm (quarter notes per minute) if no tempo information is available.

Parameters

- **music** (*muspy.Music*) – Object to inspect.
- **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

muspy.**remove_duplicate**(*obj: muspy.base.ComplexBase*) → muspy.base.ComplexBase
Remove duplicate change events.

Parameters **obj** (*muspy.Music*) – Object to process.

muspy.**sort**(*obj: muspy.base.ComplexBase*) → muspy.base.ComplexBase
Sort all the time-stamped objects with respect to event time.

- If a *muspy.Music* is given, this will sort key signatures, time signatures, lyrics and annotations, along with notes, lyrics and annotations for each track.
- If a *muspy.Track* is given, this will sort notes, lyrics and annotations.

Parameters **obj** (*muspy.ComplexBase*) – Object to sort.

muspy.**to_ordered_dict**(*obj: muspy.base.Base*, *ignore_null: bool = True*) → collections.OrderedDict
Return an OrderedDict converted from a Music object.

Parameters **obj** (*muspy.Base*) – Object to convert.

Returns Converted OrderedDict.

Return type OrderedDict

muspy.**transpose**(*obj: Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note]*, *semitone: int*) → Union[muspy.music.Music, muspy.classes.Track, muspy.classes.Note]
Transpose all the notes by a number of semitones.

Parameters

- **obj** (*muspy.Music*, *muspy.Track* or *muspy.Note*) – Object to transpose.
- **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

class muspy.**ABCFolderDataset**(*root: Union[str, pathlib.Path]*, *convert: bool = False*, *kind: str = 'json'*, *n_jobs: int = 1*, *ignore_exceptions: bool = True*, *use_converted: Optional[bool] = None*)
Class for datasets storing ABC files in a folder.

See also:

muspy.FolderDataset Class for datasets storing files in a folder.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Tuple[str, Tuple[int, int]]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**Dataset**
Base class for MusPy datasets.

To build a custom dataset, it should inherit this class and overide the methods `__getitem__` and `__len__` as well as the class attribute `_info`. `__getitem__` should return the `i`-th data sample as a *muspy.Music*

object. `__len__` should return the size of the dataset. `_info` should be a *muspy.DatasetInfo* instance storing the dataset information.

classmethod citation()
Print the citation infomation.

classmethod info()
Return the dataset infomation.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)
Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]
Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None, representation: Optional[str] = None, split_filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None, **kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]
Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.

- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to numpy.random.RandomState, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to numpy.random.RandomState, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

class muspy.**DatasetInfo**(*name: Optional[str] = None*, *description: Optional[str] = None*, *homepage: Optional[str] = None*, *license: Optional[str] = None*)

A container for dataset information.

class muspy.**EssenFolkSongDatabase**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Essen Folk Song Database.

class muspy.**FolderDataset**(*root: Union[str, pathlib.Path], convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Class for datasets storing files in a folder.

This class extends *muspy.Dataset* to support folder datasets. To build a custom folder dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set class attribute `_extension` to the extension to look for when building the dataset and set `read` to a callable that takes as inputs a filename of a source file and return the converted Music object.

root

Root directory of the dataset.

Type str or Path

Parameters

- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

---

Important: *muspy.FolderDataset.converted_exists()* depends solely on a special file named `.muspy.success` in the folder `{root}/_converted/`, which serves as an indicator for the existence and integrity of the converted dataset. If the converted dataset is built by *muspy.FolderDataset.convert()*, the `.muspy.success` file will be created as well. If the converted dataset is created manually, make sure to create the `.muspy.success` file in the folder `{root}/_converted/` to prevent errors.

Notes

Two modes are available for this dataset. When the on-the-fly mode is enabled, a data sample is converted to a music object on the fly when being indexed. When the on-the-fly mode is disabled, a data sample is loaded from the precomputed converted data.

See also:

muspy.Dataset Base class for MusPy datasets.

convert(*kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*) → FolderDatasetType

Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Returns

Return type Object itself.

converted_dir
Path to the root directory of the converted dataset.

converted_exists() → bool
Return True if the saved dataset exists, otherwise False.

exists() → bool
Return True if the dataset exists, otherwise False.

load(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Load a file into a Music object.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Any*) → muspy.music.Music
Read a file into a Music object.

use_converted() → FolderDatasetType
Disable on-the-fly mode and use converted data.

Returns

Return type Object itself.

class muspy.**HymnalDataset**(*root: Union[str, pathlib.Path], download: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Hymnal Dataset.

download() → muspy.datasets.base.FolderDataset
Download the source datasets.

Returns

Return type Object itself.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**HymnalTuneDataset**(*root: Union[str, pathlib.Path], download: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Hymnal Dataset (tune only).

download() → muspy.datasets.base.FolderDataset
Download the source datasets.

Returns

Return type Object itself.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**JSBChoralesDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Johann Sebastian Bach Chorales Dataset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**LakhMIDIAlignedDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Lakh MIDI Dataset - aligned subset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**LakhMIDIDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Lakh MIDI Dataset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**LakhMIDIMatchedDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Lakh MIDI Dataset - matched subset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**MAESTRODatasetV1**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
MAESTRO Dataset (MIDI only).

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**MAESTRODatasetV2**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
MAESTRO Dataset (MIDI only).

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**Music21Dataset**(*composer: Optional[str] = None*)
A class of datasets containing files in music21 corpus.

Parameters

- **composer** (*str*) – Name of a composer or a collection. Please refer to the music21 corpus reference page for a full list [1].
- **extensions** (*list of str*) – File extensions of desired files.

References

[1] https://web.mit.edu/music21/doc/about/referenceCorpus.html

convert(*root: Union[str, pathlib.Path], kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*) → muspy.datasets.base.MusicDataset
Convert and save the Music objects.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

class muspy.**MusicDataset**(*root: Union[str, pathlib.Path], kind: str = 'json'*)
Class for datasets of MusPy JSON/YAML files.

root
Root directory of the dataset.

Type str or Path

kind
File format of the data. Defaults to 'json'.

Type {'json', 'yaml'}, optional

See also:

muspy.Dataset Base class for MusPy datasets.

class muspy.**NESMusicDatabase**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
NES Music Database.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.**NottinghamDatabase**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Nottingham Database.

class muspy.**RemoteABCFolderDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Base class for remote datasets storing ABC files in a folder.

See also:

muspy.ABCFolderDataset Class for datasets storing ABC files in a folder.

muspy.RemoteDataset Base class for remote MusPy datasets.

class muspy.**RemoteDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False*)

Base class for remote MusPy datasets.

This class extends *muspy.Dataset* to support remote datasets. To build a custom remote dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set the class attribute `_sources` to the URLs to the source files (see Notes).

root

Root directory of the dataset.

Type str or Path

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.

Raises RuntimeError: – If `download_and_extract` is False but file `{root}/.muspy.success` does not exist (see below).

Important: `muspy.Dataset.exists()` depends solely on a special file named `.muspy.success` in directory `{root}/_converted/`. This file serves as an indicator for the existence and integrity of the dataset. It will automatically be created if the dataset is successfully downloaded and extracted by `muspy.Dataset.download_and_extract()`. If the dataset is downloaded manually, make sure to create the `.muspy.success` file in directory `{root}/_converted/` to prevent errors.

Notes

The class attribute `_sources` is a dictionary storing the following information of each source file.

- filename (str): Name to save the file.
- url (str): URL to the file.
- archive (bool): Whether the file is an archive.

- md5 (str, optional): Expected MD5 checksum of the file.

Here is an example.:

```
_sources = {
    "example": {
        "filename": "example.tar.gz",
        "url": "https://www.example.com/example.tar.gz",
        "archive": True,
        "md5": None,
    }
}
```

See also:

muspy.Dataset Base class for MusPy datasets.

download() → RemoteDatasetType
Download the source datasets.

Returns

Return type Object itself.

download_and_extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archives.

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

Notes

Equivalent to `RemoteDataset.download().extract(cleanup)`.

exists() → bool
Return True if the dataset exists, otherwise False.

extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archive(s).

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

source_exists() → bool
Return True if all the sources exist, otherwise False.

class muspy.**RemoteFolderDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Base class for remote datasets stoting files in a folder.

root
Root directory of the dataset.

Type str or Path

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.
- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

See also:

muspy.FolderDataset Class for datasets storing files in a folder.

muspy.RemoteDataset Base class for remote MusPy datasets.

read(*filename: str*) → muspy.music.Music
Read a file into a Music object.

class muspy.**RemoteMusicDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, kind: str = 'json'*)
Base class for remote datasets of MusPy JSON/YAML files.

root
Root directory of the dataset.

Type str or Path

kind
File format of the data. Defaults to 'json'.

Type {'json', 'yaml'}, optional

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.

See also:

muspy.MusicDataset Class for datasets of MusPy JSON/YAML files.

muspy.RemoteDataset Base class for remote MusPy datasets.

class muspy.**WikifoniaDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Wikifonia dataset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music

Read a file into a Music object.

muspy.**get_dataset**(*key: str*) → Type[muspy.datasets.base.Dataset]

Return a certain dataset class by key.

Parameters **key** (*str*) – Dataset key (case-insensitive).

Returns

Return type The corresponding dataset class.

muspy.**list_datasets**()

Return all supported dataset classes as a list.

Returns

Return type A list of all supported dataset classes.

muspy.**download_bravura_font**()

Download the Bravura font.

muspy.**download_musescore_soundfont**()

Download the MuseScore General soundfont.

muspy.**get_bravura_font_dir**() → pathlib.Path

Return path to the directory of the Bravura font.

muspy.**get_bravura_font_path**() → pathlib.Path

Return path to the Bravura font.

muspy.**get_musescore_soundfont_dir**() → pathlib.Path

Return path to the MuseScore General soundfont directory.

muspy.**get_musescore_soundfont_path**() → pathlib.Path

Return path to the MuseScore General soundfont.

exception muspy.**MIDIError**

An error class for MIDI related exceptions.

exception muspy.**MusicXMLError**

An error class for MusicXML related exceptions.

muspy.**from_event_representation**(*array: numpy.ndarray, resolution: int = 24, program: int = 0, is_drum: bool = False, use_single_note_off_event: bool = False, use_end_of_sequence_event: bool = False, max_time_shift: int = 100, velocity_bins: int = 32, default_velocity: int = 64*) → muspy.music.Music

Decode event-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in event-based representation to decode. Will be casted to integer if not of integer type.

- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_single_note_off_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, a note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use_end_of_sequence_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **max_time_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity_bins** (*int*) – Number of velocity bins to use. Defaults to 32.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.**from_mido**(*midi: mido.midifiles.midifiles.MidiFile*, *duplicate_note_mode: str = 'fifo'*) → muspy.music.Music

Return a mido MidiFile object as a Music object.

Parameters

- **midi** (mido.MidiFile) – Mido MidiFile object to convert.
- **duplicate_note_mode** (*{'fifo', 'lifo, 'close_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close_all': close all note on messages

Returns Converted Music object.

Return type *muspy.Music*

muspy.**from_music21**(*stream: music21.stream.Stream*, *resolution=24*) → Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]

Return a music21 Stream object as Music or Track object(s).

Parameters

- **stream** (*music21.stream.Stream*) – Stream object to convert.

- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music or Track object(s).

Return type *muspy.Music* or *muspy.Track*

muspy.**from_music21_opus**(*opus: music21.stream.Opus*, *resolution=24*) → List[muspy.music.Music]

Return a music21 Opus object as a list of Music objects.

Parameters

- **opus** (*music21.stream.Opus*) – Opus object to convert.
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object.

Return type *muspy.Music*

muspy.**from_note_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_start_end: bool = False*, *encode_velocity: bool = True*, *default_velocity: int = 64*) → muspy.music.Music

Decode note-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in note-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_start_end** (*bool*) – Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **default_velocity** (*int*) – Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.**from_object**(*obj: Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty_midi.pretty_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack], **kwargs*) → Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]

Return an outside object as a Music object.

Parameters **obj** – Object to convert. Supported objects are *music21.Stream*, `mido.MidiTrack`, `pretty_midi.PrettyMIDI`, and `pypianoroll.Multitrack` objects.

Returns Converted Music object.

Return type *muspy.Music*

muspy.**from_pianoroll_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *encode_velocity: bool = True*, *default_velocity: int = 64*) → muspy.music.Music

Decode pitch-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in piano-roll representation to decode. Will be casted to integer if not of integer type. If *encode_velocity* is True, will be casted to boolean if not of boolean type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode velocities. Defaults to True.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.**from_pitch_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_hold_state: bool = False*, *default_velocity: int = 64*) → muspy.music.Music

Decode pitch-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in pitch-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_hold_state** (*bool*) – Whether to use a special state for holds. Defaults to False.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.**from_pretty_midi**(*midi: pretty_midi.pretty_midi.PrettyMIDI*) → muspy.music.Music
Return a pretty_midi PrettyMIDI object as a Music object.

Parameters **midi** (pretty_midi.PrettyMIDI) – PrettyMIDI object to convert.

Returns Converted Music object.

Return type *muspy.Music*

muspy.**from_pypianoroll**(*multitrack: pypianoroll.multitrack.Multitrack*, *default_velocity: int = 64*) → muspy.music.Music
Return a Pypianoroll Multitrack object as a Music object.

Parameters

- **multitrack** (pypianoroll.Multitrack) – Pypianoroll Multitrack object to convert.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns **music** – Converted MusPy Music object.

Return type *muspy.Music*

muspy.**from_representation**(*array: numpy.ndarray*, *kind: str*, ***kwargs*) → muspy.music.Music
Update with the given representation.

Parameters

- **array** (numpy.ndarray) – Array in a supported representation.
- **kind** (*str, {'pitch', 'pianoroll', 'event', 'note'}*) – Data representation type (case-insensitive).

Returns Converted Music object.

Return type *muspy.Music*

muspy.**load**(*path: Union[str, pathlib.Path]*, *kind: Optional[str] = None*, ***kwargs*) → muspy.music.Music
Load a JSON or a YAML file into a Music object.

Parameters

- **path** (*str or Path*) – Path to the file to load.
- **kind** (*{'json', 'yaml'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.
- ****kwargs** (*dict*) – Keyword arguments to pass to the target function. See *muspy.load_json()* or *muspy.load_yaml()* for available arguments.

Returns Loaded Music object.

Return type *muspy.Music*

See also:

muspy.read() Read a MIDI/MusicXML/ABC file into a Music object.

muspy.**load_json**(*path: Union[str, pathlib.Path]*) → muspy.music.Music
Load a JSON file into a Music object.

Parameters **path** (*str or Path*) – Path to the file to load.

Returns Loaded Music object.

Return type *muspy.Music*

muspy.**load_yaml**(*path: Union[str, pathlib.Path]*) → muspy.music.Music
Load a YAML file into a Music object.

Parameters **path** (*str or Path*) – Path to the file to load.

Returns Loaded Music object.

Return type *muspy.Music*

muspy.**read**(*path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs*) → Union[muspy.music.Music, List[muspy.music.Music]]
Read a MIDI/MusicXML/ABC file into a Music object.

Parameters

- **path** (*str or Path*) – Path to the file to read.
- **kind** (*{'midi', 'musicxml', 'abc'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.

Returns Converted Music object(s).

Return type *muspy.Music* or list of *muspy.Music*

See also:

muspy.load() Load a JSON or a YAML file into a Music object.

muspy.**read_abc**(*path: Union[str, pathlib.Path], number: Optional[int] = None, resolution=24*) → List[muspy.music.Music]
Return an ABC file into Music object(s) using music21 backend.

Parameters

- **path** (*str or Path*) – Path to the ABC file to read.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object(s).

Return type list of *muspy.Music*

muspy.**read_abc_string**(*data_str: str*, *number: Optional[int] = None*, *resolution=24*)
Read ABC data into Music object(s) using music21 backend.

Parameters

- **data_str** (*str*) – ABC data to parse.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object(s).

Return type *muspy.Music*

muspy.**read_midi**(*path: Union[str, pathlib.Path], backend: str = 'mido', duplicate_note_mode: str = 'fifo'*) → muspy.music.Music

Read a MIDI file into a Music object.

Parameters

- **path** (*str or Path*) – Path to the MIDI file to read.
- **backend** (*{'mido', 'pretty_midi'}*) – Backend to use.
- **duplicate_note_mode** (*{'fifo', 'lifo, 'close_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'. Only used when *backend='mido'*.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close_all': close all note on messages

Returns Converted Music object.

Return type *muspy.Music*

muspy.**read_musicxml**(*path: Union[str, pathlib.Path], compressed: Optional[bool] = None*) → muspy.music.Music

Read a MusicXML file into a Music object.

Parameters **path** (*str or Path*) – Path to the MusicXML file to read.

Returns Converted Music object.

Return type *muspy.Music*

Notes

Grace notes and unpitched notes are not supported.

muspy.**drum_in_pattern_rate**(*music: muspy.music.Music, meter: str*) → float

Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$drum\_in\_pattern\_rate = \frac{\#(drum\_notes\_in\_pattern)}{\#(drum\_notes)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **meter** (*str, {'duple', 'triple'}*) – Meter of the drum pattern.

Returns Drum-in-pattern rate.

Return type float

See also:

muspy.drum_pattern_consistency() Compute the largest drum-in-pattern rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**drum_pattern_consistency**(*music: muspy.music.Music*) → float

Return the largest drum-in-pattern rate.

The drum pattern consistency is defined as the largest drum-in-pattern rate over duple and triple meters. Only drum tracks are considered. Return NaN if no drum note is found.

$$drum\_pattern\_consistency = \max_{meter} drum\_in\_pattern\_rate(meter)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Drum pattern consistency.

Return type float

See also:

muspy.drum_in_pattern_rate() Compute the ratio of drum notes in a certain drum pattern.

muspy.**empty_beat_rate**(*music: muspy.music.Music*) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero. This metric is also implemented in Pypianoroll [1].

$$empty\_beat\_rate = \frac{\#(empty\_beats)}{\#(beats)}$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Empty-beat rate.

Return type float

See also:

muspy.empty_measure_rate() Compute the ratio of empty measures.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, "Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls," in Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2018.

muspy.**empty_measure_rate**(*music: muspy.music.Music*, *measure_resolution: int*) → float

Return the ratio of empty measures.

The empty-measure rate is defined as the ratio of the number of empty measures (where no note is played) to the total number of measures. Note that this metric only works for songs with a constant time signature. Return NaN if song length is zero. This metric is used in [1].

$$empty\_measure\_rate = \frac{\#(empty\_measures)}{\#(measures)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure_resolution** (*int*) – Time steps per measure.

Returns Empty-measure rate.

Return type float

See also:

muspy.empty_beat_rate() Compute the ratio of empty beats.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**groove_consistency**(*music: muspy.music.Music*, *measure_resolution: int*) → float

Return the groove consistency.

The groove consistency is defined as the mean hamming distance of the neighboring measures.

$$groove\_consistency = 1 - \frac{1}{T-1}\sum_{i=1}^{T-1} d(G_i, G_{i+1})$$

Here, T is the number of measures, G_i is the binary onset vector of the i-th measure (a one at position that has an onset, otherwise a zero), and $d(G, G')$ is the hamming distance between two vectors G and G'. Note that this metric only works for songs with a constant time signature. Return NaN if the number of measures is less than two. This metric is used in [1].

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure_resolution** (*int*) – Time steps per measure.

Returns Groove consistency.

Return type float

References

1. Shih-Lun Wu and Yi-Hsuan Yang, "The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures", in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

muspy.**n_pitch_classes_used**(*music: muspy.music.Music*) → int

Return the number of unique pitch classes used.

Drum tracks are ignored.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Number of unique pitch classes used.

Return type int

See also:

muspy.n_pitches_used() Compute the number of unique pitches used.

muspy.**n_pitches_used**(*music: muspy.music.Music*) → int
Return the number of unique pitches used.

Drum tracks are ignored.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Number of unique pitch used.

Return type int

See also:

muspy.n_pitch_class_used() Compute the number of unique pitch classes used.

muspy.**pitch_class_entropy**(*music: muspy.music.Music*) → float
Return the entropy of the normalized note pitch class histogram.

The pitch class entropy is defined as the Shannon entropy of the normalized note pitch class histogram. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_class\_entropy = -\sum_{i=0}^{11} P(pitch\_class = i) \times \log_2 P(pitch\_class = i)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch class entropy.

Return type float

See also:

muspy.pitch_entropy() Compute the entropy of the normalized pitch histogram.

References

1. Shih-Lun Wu and Yi-Hsuan Yang, "The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures", in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

muspy.**pitch_entropy**(*music: muspy.music.Music*) → float
Return the entropy of the normalized note pitch histogram.

The pitch entropy is defined as the Shannon entropy of the normalized note pitch histogram. Drum tracks are ignored. Return NaN if no note is found.

$$pitch\_entropy = -\sum_{i=0}^{127} P(pitch = i) \log_2 P(pitch = i)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch entropy.

Return type float

See also:

muspy.pitch_class_entropy() Compute the entropy of the normalized pitch class histogram.

muspy.**pitch_in_scale_rate**(*music: muspy.music.Music*, *root: int*, *mode: str*) → float
Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_in\_scale\_rate = \frac{\#(notes\_in\_scale)}{\#(notes)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str, {'major', 'minor'}*) – Mode of the scale.

Returns Pitch-in-scale rate.

Return type float

See also:

muspy.scale_consistency() Compute the largest pitch-in-class rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**pitch_range**(*music: muspy.music.Music*) → int
Return the pitch range.

Drum tracks are ignored. Return zero if no note is found.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch range.

Return type int

muspy.**polyphony**(*music: muspy.music.Music*) → float
Return the average number of pitches being played concurrently.

The polyphony is defined as the average number of pitches being played at the same time, evaluated only at time steps where at least one pitch is on. Drum tracks are ignored. Return NaN if no note is found.

$$polyphony = \frac{\#(pitches\_when\_at\_least\_one\_pitch\_is\_on)}{\#(time\_steps\_where\_at\_least\_one\_pitch\_is\_on)}$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Polyphony.

Return type float

See also:

muspy.polyphony_rate() Compute the ratio of time steps where multiple pitches are on.

muspy.**polyphony_rate**(*music: muspy.music.Music*, *threshold: int = 2*) → float

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$polyphony\_rate = \frac{\#(time\_steps\_where\_multiple\_pitches\_are\_on)}{\#(time\_steps)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

Returns Polyphony rate.

Return type float

See also:

muspy.polyphony() Compute the average number of pitches being played at the same time.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.**scale_consistency**(*music: muspy.music.Music*) → float

Return the largest pitch-in-scale rate.

The scale consistency is defined as the largest pitch-in-scale rate over all major and minor scales. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$scale\_consistency = \max_{root,mode} pitch\_in\_scale\_rate(root, mode)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Scale consistency.

Return type float

See also:

muspy.pitch_in_scale_rate() Compute the ratio of pitches in a certain musical scale.

References

1. Olof Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training," in NeuIPS Workshop on Constructive Machine Learning, 2016.

class muspy.**Music**(*metadata: Optional[muspy.classes.Metadata] = None*, *resolution: Optional[int] = None*, *tempos: Optional[List[muspy.classes.Tempo]] = None*, *key_signatures: Optional[List[muspy.classes.KeySignature]] = None*, *time_signatures: Optional[List[muspy.classes.TimeSignature]] = None*, *downbeats: Optional[List[int]] = None*, *lyrics: Optional[List[muspy.classes.Lyric]] = None*, *annotations: Optional[List[muspy.classes.Annotation]] = None*, *tracks: Optional[List[muspy.classes.Track]] = None*)

A universal container for symbolic music.

This is the core class of MusPy. A Music object can be constructed in the following ways.

- *muspy.Music()*: Construct by setting values for attributes.
- muspy.Music.from_dict(): Construct from a dictionary that stores the attributes and their values as key-value pairs.
- *muspy.read()*: Read from a MIDI, a MusicXML or an ABC file.
- *muspy.load()*: Load from a JSON or a YAML file saved by *muspy.save()*.
- *muspy.from_object()*: Convert from a *music21.Stream*, a mido.MidiFile, a pretty_midi.PrettyMIDI or a pypianoroll.Multitrack object.

metadata
Metadata.

Type *muspy.Metadata*

resolution
Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Type int, optional

tempos
Tempo changes.

Type list of *muspy.Tempo*

key_signatures
Key signatures changes.

Type list of *muspy.KeySignature*

time_signatures
Time signature changes.

Type list of *muspy.TimeSignature*

downbeats
Downbeat positions.

Type list of int

lyrics
Lyrics.

Type list of *muspy.Lyric*

annotations
Annotations.

Type list of *muspy.Annotation*

tracks
Music tracks.

Type list of *muspy.Track*

Note: Indexing a Music object returns the track of a certain index. That is, `music[idx]` returns `music.tracks[idx]`. Length of a Music object is the number of tracks. That is, `len(music)` returns `len(music.tracks)`.

---

adjust_resolution(*target: Optional[int] = None*, *factor: Optional[float] = None*, *rounding: Union[str, Callable, None] = 'round'*) → muspy.music.Music
Adjust resolution and timing of all time-stamped objects.

Parameters

- **target** (*int, optional*) – Target resolution.
- **factor** (*int or float, optional*) – Factor used to adjust the resolution based on the formula: *new_resolution = old_resolution * factor*. For example, a factor of 2 double the resolution, and a factor of 0.5 halve the resolution.
- **rounding** (*{'round', 'ceil', 'floor'} or callable, optional*) – Rounding mode. Defaults to 'round'.

Returns

Return type Object itself.

clip(*lower: int = 0*, *upper: int = 127*) → muspy.music.Music
Clip the velocity of each note for each track.

Parameters

- **lower** (*int, optional*) – Lower bound. Defaults to 0.
- **upper** (*int, optional*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

get_end_time(*is_sorted: bool = False*) → int
Return the the time of the last event in all tracks.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations.

Parameters **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

get_real_end_time(*is_sorted: bool = False*) → float
Return the end time in realtime.

This includes tempos, key signatures, time signatures, note offsets, lyrics and annotations. Assume 120 qpm (quarter notes per minute) if no tempo information is available.

Parameters **is_sorted** (*bool*) – Whether all the list attributes are sorted. Defaults to False.

save(*path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs*)
Save loselessly to a JSON or a YAML file.

Refer to *muspy.save()* for full documentation.

save_json(*path: Union[str, pathlib.Path], **kwargs*)
Save loselessly to a JSON file.

Refer to *muspy.save_json()* for full documentation.

save_yaml (*path: Union[str, pathlib.Path]*)
Save loselessly to a YAML file.

Refer to *muspy.save_yaml()* for full documentation.

show (*kind: str*, ***kwargs*)
Show visualization.

Refer to *muspy.show()* for full documentation.

show_pianoroll (***kwargs*)
Show pianoroll visualization.

Refer to *muspy.show_pianoroll()* for full documentation.

show_score (***kwargs*)
Show score visualization.

Refer to *muspy.show_score()* for full documentation.

synthesize (***kwargs*) → numpy.ndarray
Synthesize a Music object to raw audio.

Refer to *muspy.synthesize()* for full documentation.

to_event_representation (***kwargs*) → numpy.ndarray
Return in event-based representation.

Refer to *muspy.to_event_representation()* for full documentation.

to_mido (***kwargs*) → mido.midifiles.midifiles.MidiFile
Return as a MidiFile object.

Refer to *muspy.to_mido()* for full documentation.

to_music21 (***kwargs*) → music21.stream.Stream
Return as a Stream object.

Refer to *muspy.to_music21()* for full documentation.

to_note_representation (***kwargs*) → numpy.ndarray
Return in note-based representation.

Refer to *muspy.to_note_representation()* for full documentation.

to_object (*kind: str*, ***kwargs*)
Return as an object in other libraries.

Refer to *muspy.to_object()* for full documentation.

to_pianoroll_representation (***kwargs*) → numpy.ndarray
Return in piano-roll representation.

Refer to *muspy.to_pianoroll_representation()* for full documentation.

to_pitch_representation (***kwargs*) → numpy.ndarray
Return in pitch-based representation.

Refer to *muspy.to_pitch_representation()* for full documentation.

to_pretty_midi (***kwargs*) → pretty_midi.pretty_midi.PrettyMIDI
Return as a PrettyMIDI object.

Refer to *muspy.to_pretty_midi()* for full documentation.

to_pypianoroll(***kwargs*) → pypianoroll.multitrack.Multitrack
Return as a Multitrack object.

Refer to *muspy.to_pypianoroll()* for full documentation.

to_representation(*kind: str*, ***kwargs*) → numpy.ndarray
Return in a specific representation.

Refer to *muspy.to_representation()* for full documentation.

transpose(*semitone: int*) → muspy.music.Music
Transpose all the notes by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose the notes. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

Notes

Drum tracks are skipped.

write(*path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs*)
Write to a MIDI, a MusicXML, an ABC or an audio file.

Refer to *muspy.write()* for full documentation.

write_abc(*path: Union[str, pathlib.Path], **kwargs*)
Write to an ABC file.

Refer to *muspy.write_abc()* for full documentation.

write_audio(*path: Union[str, pathlib.Path], **kwargs*)
Write to an audio file.

Refer to *muspy.write_audio()* for full documentation.

write_midi(*path: Union[str, pathlib.Path], **kwargs*)
Write to a MIDI file.

Refer to *muspy.write_midi()* for full documentation.

write_musicxml(*path: Union[str, pathlib.Path], **kwargs*)
Write to a MusicXML file.

Refer to *muspy.write_musicxml()* for full documentation.

muspy.**save**(*path: Union[str, pathlib.Path], music: Music, kind: Optional[str] = None, **kwargs*)
Save a Music object loselessly to a JSON or a YAML file.

Parameters

- **path** (*str or Path*) – Path to save the file.
- **music** (*muspy.Music*) – Music object to save.
- **kind** (*{'json', 'yaml'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.

See also:

muspy.write() Write a Music object to a MIDI/MusicXML/ABC/audio file.

Notes

The conversion can be lossy if any nonserializable object is used (for example, an Annotation object, which can store data of any type).

muspy.**save_json**(*path: Union[str, pathlib.Path]*, *music: Music*)
Save a Music object to a JSON file.

Parameters

- **path** (*str or Path*) – Path to save the JSON file.
- **music** (*muspy.Music*) – Music object to save.

muspy.**save_yaml**(*path: Union[str, pathlib.Path]*, *music: Music*)
Save a Music object to a YAML file.

Parameters

- **path** (*str or Path*) – Path to save the YAML file.
- **music** (*muspy.Music*) – Music object to save.

muspy.**to_event_representation**(*music: Music*, *use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *encode_velocity: bool = False*, *force_velocity_event: bool = True*, *max_time_shift: int = 100*, *velocity_bins: int = 32*) → numpy.ndarray
Encode a Music object into event-based representation.

The event-based represetantion represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is M x 1, where M is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-one events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_single_note_off_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use_end_of_sequence_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode velocities.
- **force_velocity_event** (*bool*) – Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.
- **max_time_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity_bins** (*int*) – Number of velocity bins to use. Defaults to 32.

Returns Encoded array in event-based representation.

Return type ndarray, dtype=uint16, shape=(?, 1)

muspy.**to_mido**(*music: Music*, *use_note_on_as_note_off: bool = True*)
Return a Music object as a MidiFile object.

Parameters

- **music** (*muspy.Music* object) – Music object to convert.
- **use_note_on_as_note_off** (*bool*) – Whether to use a note on message with zero velocity instead of a note off message.

Returns Converted MidiFile object.

Return type mido.MidiFile

muspy.**to_music21**(*music: Music*) → music21.stream.Score

Return a Music object as a music21 Score object.

Parameters **music** (*muspy.Music*) – Music object to convert.

Returns Converted music21 Score object.

Return type *music21.stream.Score*

muspy.**to_note_representation**(*music: Music*, *use_start_end: bool = False*, *encode_velocity: bool = True*, *dtype: Union[numpy.dtype, type, str] = <class 'int'>*) → numpy.ndarray

Encode a Music object into note-based representation.

The note-based represetantion represents music as a sequence of (time, pitch, duration, velocity) tuples. For example, a note Note(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 60, 4, 64). The output shape is N * D, where N is the number of notes and D is 4 when *encode_velocity* is True, otherwise D is 3. The values of the second dimension represent time, pitch, duration and velocity (discarded when *encode_velocity* is False).

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_start_end** (*bool*) – Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **dtype** (*dtype, type or str*) – Data type of the return array. Defaults to int.

Returns Encoded array in note-based representation.

Return type ndarray, shape=(?, 3 or 4)

muspy.**to_object**(*music: Music*, *kind: str*, ***kwargs*) → Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty_midi.pretty_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack]

Return a Music object as an object in other libraries.

Supported classes are *music21.Stream*, mido.MidiTrack, pretty_midi.PrettyMIDI and pypianoroll.Multitrack.

Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str, {'music21', 'mido', 'pretty_midi', 'pypianoroll'}*) – Target class (case-insensitive).

Returns Converted object.

Return type *music21.Stream*, mido.MidiTrack, pretty_midi.PrettyMIDI or pypianoroll.Multitrack

muspy.**to_pianoroll_representation**(*music: Music*, *encode_velocity: bool = True*) → numpy.ndarray

Encode notes into piano-roll representation.

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **encode_velocity** (*bool*) – Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

Returns Encoded array in piano-roll representation.

Return type ndarray, dtype=uint8 or bool, shape=(?, 128)

muspy.**to_pitch_representation**(*music: Music*, *use_hold_state: bool = False*) → numpy.ndarray

Encode a Music object into pitch-based representation.

The pitch-based represetantion represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_hold_state** (*bool*) – Whether to use a special state for holds. Defaults to False.

Returns Encoded array in pitch-based representation.

Return type ndarray, dtype=uint8, shape=(?, 1)

muspy.**to_pretty_midi**(*music: Music*) → pretty_midi.pretty_midi.PrettyMIDI

Return a Music object as a PrettyMIDI object.

Tempo changes are not supported yet.

Parameters **music** (*muspy.Music* object) – Music object to convert.

Returns Converted PrettyMIDI object.

Return type pretty_midi.PrettyMIDI

muspy.**to_pypianoroll**(*music: Music*) → pypianoroll.multitrack.Multitrack

Return a Music object as a Multitrack object.

Parameters **music** (*muspy.Music*) – Music object to convert.

Returns **multitrack** – Converted Multitrack object.

Return type pypianoroll.Multitrack

muspy.**to_representation**(*music: Music*, *kind: str*, ***kwargs*) → numpy.ndarray

Return a Music object in a specific representation.

Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str, {'pitch', 'piano-roll', 'event', 'note'}*) – Target representation (case-insensitive).

Returns **array** – Converted representation.

Return type ndarray

muspy.**synthesize**(*music: Music*, *soundfont_path: Union[str, pathlib.Path, None] = None*, *rate: int = 44100*) → numpy.ndarray

Synthesize a Music object to raw audio.

Parameters

- **music** (*muspy.Music*) – Music object to write.
- **soundfont_path** (*str or Path, optional*) – Path to the soundfount file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.

Returns Synthesized waveform.

Return type ndarray, dtype=int16, shape=(?, 2)

muspy.**write**(*path: Union[str, pathlib.Path], music: Music, kind: Optional[str] = None, **kwargs*)

Write a Music object to a MIDI/MusicXML/ABC/audio file.

Parameters

- **path** (*str or Path*) – Path to write the file.
- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*{'midi', 'musicxml', 'abc', 'audio'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.

See also:

muspy.save() Save a Music object loselessly to a JSON or a YAML file.

muspy.**write_abc**(*path: Union[str, pathlib.Path], music: Music*)

Write a Music object to a ABC file.

Parameters

- **path** (*str or Path*) – Path to write the ABC file.
- **music** (*muspy.Music*) – Music object to write.

muspy.**write_audio**(*path: Union[str, pathlib.Path], music: Music, soundfont_path: Union[str, pathlib.Path, None] = None, rate: int = 44100, audio_format: Optional[str] = None*)

Write a Music object to an audio file.

Supported formats include WAV, AIFF, FLAC and OGA.

Parameters

- **path** (*str or Path*) – Path to write the audio file.
- **music** (*muspy.Music*) – Music object to write.
- **soundfont_path** (*str or Path, optional*) – Path to the soundfount file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.
- **audio_format** (*str, {'wav', 'aiff', 'flac', 'oga'}, optional*) – File format to write. If None, infer it from the extension.

muspy.**write_midi**(*path: Union[str, pathlib.Path], music: Music, backend: str = 'mido', **kwargs*)

Write a Music object to a MIDI file.

Parameters

- **path** (*str or Path*) – Path to write the MIDI file.
- **music** (*muspy.Music*) – Music object to write.
- **backend** (*{'mido', 'pretty_midi'}*) – Backend to use. Defaults to 'mido'.

muspy.**write_musicxml**(*path: Union[str, pathlib.Path], music: Music, compressed: Optional[bool] = None*)

Write a Music object to a MusicXML file.

Parameters

- **path** (*str or Path*) – Path to write the MusicXML file.
- **music** (*muspy.Music*) – Music object to write.
- **compressed** (*bool, optional*) – Whether to write to a compressed MusicXML file. If None, infer from the extension of the filename ('.xml' and '.musicxml' for an uncompressed file, '.mxl' for a compressed file).

class muspy.**NoteRepresentationProcessor**(*use_start_end: bool = False, encode_velocity: bool = True, dtype: Union[numpy.dtype, type, str] = <class 'int'>, default_velocity: int = 64*)

Note-based representation processor.

The note-based represetantion represents music as a sequence of (pitch, time, duration, velocity) tuples. For example, a note Note(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 4, 60, 64). The output shape is L * D, where L is th number of notes and D is 4 when *encode_velocity* is True, otherwise D is 3. The values of the second dimension represent pitch, time, duration and velocity (discarded when *encode_velocity* is False).

use_start_end
Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.

Type bool

encode_velocity
Whether to encode note velocities. Defaults to True.

Type bool

dtype
Data type of the return array. Defaults to int.

Type dtype, type or str

default_velocity
Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode note-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in note-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_note_representation() Return a Music object converted from note-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into note-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in note-based representation.

Return type ndarray (np.uint8)

See also:

muspy.to_note_representation() Convert a Music object into note-based representation.

class muspy.**EventRepresentationProcessor**(*use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *encode_velocity: bool = False*, *force_velocity_event: bool = True*, *max_time_shift: int = 100*, *velocity_bins: int = 32*, *default_velocity: int = 64*)
Event-based representation processor.

The event-based represetantion represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is M x 1, where M is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-one events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

use_single_note_off_event
Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.

Type bool

use_end_of_sequence_event
Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.

Type bool

encode_velocity
Whether to encode velocities.

Type bool

force_velocity_event
Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

Type bool

max_time_shift
Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.

Type int

velocity_bins
Number of velocity bins to use. Defaults to 32.

Type int

default_velocity
Default velocity value to use when decoding. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode event-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_event_representation() Return a Music object converted from event-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into event-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in event-based representation.

Return type ndarray (np.uint16)

See also:

muspy.to_event_representation() Convert a Music object into event-based representation.

class muspy.**PianoRollRepresentationProcessor**(*encode_velocity: bool = True*, *default_velocity: int = 64*)
Piano-roll representation processor.

The piano-roll represetantion represents music as a time-pitch matrix, where the columns are the time steps and the rows are the pitches. The values indicate the presence of pitches at different time steps. The output shape is T x 128, where T is the number of time steps.

encode_velocity
Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

Type bool

default_velocity
Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode piano-roll representation into a Music object.

Parameters **array** (*ndarray*) – Array in piano-roll representation to decode. Cast to integer if not of integer type. If *encode_velocity* is True, casted to boolean if not of boolean type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_pianoroll_representation() Return a Music object converted from piano-roll representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into piano-roll representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in piano-roll representation.

Return type ndarray (np.uint8)

See also:

muspy.to_pianoroll_representation() Convert a Music object into piano-roll representation.

class muspy.**PitchRepresentationProcessor**(*use_hold_state: bool = False*, *default_velocity: int = 64*)
Pitch-based representation processor.

The pitch-based represetantion represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

use_hold_state
Whether to use a special state for holds. Defaults to False.

Type bool

default_velocity
Default velocity value to use when decoding. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode pitch-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in pitch-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_pitch_representation() Return a Music object converted from pitch-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into pitch-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in pitch-based representation.

Return type ndarray (np.uint8)

See also:

muspy.to_pitch_representation() Convert a Music object into pitch-based representation.

muspy.**get_json_schema_path**() → str
Return the path to the JSON schema.

muspy.**get_musicxml_schema_path**() → str
Return the path to the MusicXML schema.

muspy.**get_yaml_schema_path**() → str
Return the path to the YAML schema.

muspy.**validate_json**(*path: Union[str, pathlib.Path]*)
Validate a file against the JSON schema.

Parameters **path** (*str or Path*) – Path to the file to validate.

muspy.**validate_musicxml**(*path: Union[str, pathlib.Path]*)
Validate a file against the MusicXML schema.

Parameters **path** (*str or Path*) – Path to the file to validate.

muspy.**validate_yaml**(*path: Union[str, pathlib.Path]*)
Validate a file against the YAML schema.

Parameters **path** (*str or Path*) – Path to the file to validate.

muspy.**show**(*music: Music*, *kind: str*, ***kwargs*)
Show visualization.

Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str, {'piano-roll', 'score'}*) – Target representation.

muspy.**show_pianoroll**(*music: Music*, ***kwargs*)
Show pianoroll visualization.

muspy.**show_score**(*music: Music*, *figsize: Optional[Tuple[float, float]] = None*, *clef: str = 'treble'*, *clef_octave: Optional[int] = 0*, *note_spacing: Optional[int] = None*, *font_path: Union[str, pathlib.Path, None] = None*, *font_scale: Optional[float] = None*) → muspy.visualization.score.ScorePlotter
Show score visualization.

Parameters

- **music** (*muspy.Music*) – Music object to show.
- **figsize** (*(float, float), optional*) – Width and height in inches. Defaults to Matplotlib configuration.
- **clef** (*str, {'treble', 'alto', 'bass'}*) – Clef type. Defaults to a treble clef.
- **clef_octave** (*int*) – Clef octave. Defaults to zero.
- **note_spacing** (*int, optional*) – Spacing of notes. Defaults to 4.
- **font_path** (*str or Path, optional*) – Path to the music font. Defaults to the path to the built-in Bravura font.
- **font_scale** (*float, optional*) – Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

Returns A ScorePlotter object that handles the score.

Return type *muspy.ScorePlotter*

class muspy.**ScorePlotter**(*fig: matplotlib.figure.Figure*, *ax: matplotlib.axes._axes.Axes*, *resolution: int*, *note_spacing: Optional[int] = None*, *font_path: Union[str, pathlib.Path, None] = None*, *font_scale: Optional[float] = None*)
A plotter that handles the score visualization.

fig
Figure object to plot the score on.

Type matplotlib.figure.Figure

axes
Axes object to plot the score on.

Type matplotlib.axes.Axes

resolution
Time steps per quarter note.

Type int

note_spacing
Spacing of notes. Defaults to 4.

Type int, optional

font_path
Path to the music font. Defaults to the path to the downloaded Bravura font.

Type str or Path, optional

font_scale
Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

Type float, optional

adjust_fonts(*scale: Optional[float] = None*)
Adjust the fonts.

plot_bar_line() → matplotlib.lines.Line2D
Plot a bar line.

plot_clef(*kind='treble', octave=0*) → matplotlib.text.Text
Plot a clef.

plot_final_bar_line() → List[matplotlib.artist.Artist]
Plot an ending bar line.

plot_key_signature(*root: int, mode: str*)
Plot a key signature. Only major and minor keys are supported.

plot_note(*time, duration, pitch*) → Optional[Tuple[List[matplotlib.text.Text], List[matplotlib.patches.Arc]]]
Plot a note.

plot_object(*obj*)
Plot an object.

plot_staffs(*start: Optional[float] = None, end: Optional[float] = None*) → List[matplotlib.lines.Line2D]
Plot the staffs.

plot_tempo(*qpm*) → List[matplotlib.artist.Artist]
Plot a tempo as a metronome mark.

plot_time_signature(*numerator: int, denominator: int*) → List[matplotlib.text.Text]
Plot a time signature.

set_baseline(*y*)
Set baseline position (the y-coordinate of the first staff line).

update_boundaries(*left: Optional[float] = None*, *right: Optional[float] = None*, *bottom: Optional[float] = None*, *top: Optional[float] = None*)

Update boundaries.

7.10.2 muspy.datasets

Dataset classes.

This module provides an easy-to-use dataset management system. Each supported dataset in MusPy comes with a class inherited from the base MusPy Dataset class. It also provides interfaces to PyTorch and TensorFlow for creating input pipelines for machine learning.

Base Classes

- ABCFolderDataset
- Dataset
- DatasetInfo
- FolderDataset
- RemoteABCFolderDataset
- RemoteDataset
- RemoteFolderDataset
- RemoteMusicDataset
- MusicDataset

Dataset Classes

- EssenFolkSongDatabase
- HymnalDataset
- HymnalTuneDataset
- JSBChoralesDataset
- LakhMIDIAlignedDataset
- LakhMIDIDataset
- LakhMIDIMatchedDataset
- MAESTRODatasetV1
- MAESTRODatasetV2
- Music21Dataset
- NESMusicDatabase
- NottinghamDatabase
- WikifoniaDataset

class muspy.datasets.**ABCFolderDataset**(*root: Union[str, pathlib.Path], convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Class for datasets storing ABC files in a folder.

See also:

muspy.FolderDataset Class for datasets storing files in a folder.

on_the_fly() → FolderDatasetType

Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Tuple[str, Tuple[int, int]]*) → muspy.music.Music

Read a file into a Music object.

class muspy.datasets.**Dataset**

Base class for MusPy datasets.

To build a custom dataset, it should inherit this class and overide the methods `__getitem__` and `__len__` as well as the class attribute `_info`. `__getitem__` should return the `i`-th data sample as a *muspy.Music* object. `__len__` should return the size of the dataset. `_info` should be a *muspy.DatasetInfo* instance storing the dataset information.

classmethod citation()

Print the citation infomation.

classmethod info()

Return the dataset infomation.

save(*root: Union[str, pathlib.Path], kind: Optional[str] = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*)

Save all the music objects to a directory.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Notes

The converted files will be named by its index. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

split(*filename: Union[str, pathlib.Path, None] = None, splits: Optional[Sequence[float]] = None, random_state: Any = None*) → Dict[str, List[int]]

Return the dataset as a PyTorch dataset.

Parameters

- **filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

to_pytorch_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TorchDataset, Dict[str, TorchDataset]]

Return the dataset as a PyTorch dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.
- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns Converted PyTorch dataset(s).

Return type class:torch.utils.data.Dataset‘ or Dict of :class:torch.utils.data.Dataset‘

to_tensorflow_dataset(*factory: Optional[Callable] = None*, *representation: Optional[str] = None*, *split_filename: Union[str, pathlib.Path, None] = None*, *splits: Optional[Sequence[float]] = None*, *random_state: Any = None*, ***kwargs*) → Union[TFDataset, Dict[str, TFDataset]]

Return the dataset as a TensorFlow dataset.

Parameters

- **factory** (*Callable, optional*) – Function to be applied to the Music objects. The input is a Music object, and the output is an array or a tensor.
- **representation** (*str, optional*) – Target representation. See *muspy.to_representation()* for available representation.
- **split_filename** (*str or Path, optional*) – If given and exists, path to the file to read the split from. If None or not exists, path to save the split.
- **splits** (*float or list of float, optional*) – Ratios for train-test-validation splits. If None, return the full dataset as a whole. If float, return train and

test splits. If list of two floats, return train and test splits. If list of three floats, return train, test and validation splits.

- **random_state** (*int, array_like or RandomState, optional*) – Random state used to create the splits. If int or array_like, the value is passed to `numpy.random.RandomState`, and the created RandomState object is used to create the splits. If RandomState, it will be used to create the splits.

Returns

- class:tensorflow.data.Dataset‘ or Dict of
- class:tensorflow.data.dataset‘ – Converted TensorFlow dataset(s).

class muspy.datasets.**DatasetInfo**(*name: Optional[str] = None, description: Optional[str] = None, homepage: Optional[str] = None, license: Optional[str] = None*)

A container for dataset information.

class muspy.datasets.**EssenFolkSongDatabase**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Essen Folk Song Database.

class muspy.datasets.**FolderDataset**(*root: Union[str, pathlib.Path], convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Class for datasets storing files in a folder.

This class extends *muspy.Dataset* to support folder datasets. To build a custom folder dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set class attribute `_extension` to the extension to look for when building the dataset and set `read` to a callable that takes as inputs a filename of a source file and return the converted Music object.

root

Root directory of the dataset.

Type str or Path

Parameters

- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to ‘json’.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

Important: *muspy.FolderDataset.converted_exists()* depends solely on a special file named `.muspy.success` in the folder `{root}/_converted/`, which serves as an indicator for the existence and integrity of the converted dataset. If the converted dataset is built by *muspy.FolderDataset.convert()*, the `.muspy.success` file will be created as well. If the converted dataset is created manually, make sure to create the `.muspy.success` file in the folder `{root}/_converted/` to prevent errors.

Notes

Two modes are available for this dataset. When the on-the-fly mode is enabled, a data sample is converted to a music object on the fly when being indexed. When the on-the-fly mode is disabled, a data sample is loaded from the precomputed converted data.

See also:

muspy.Dataset Base class for MusPy datasets.

convert(*kind: str = 'json'*, *n_jobs: int = 1*, *ignore_exceptions: bool = True*) → FolderDatasetType
Convert and save the Music objects.

The converted files will be named by its index and saved to `root/_converted`. The original filenames can be found in the `filenames` attribute. For example, the file at `filenames[i]` will be converted and saved to `{i}.json`.

Parameters

- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

Returns

Return type Object itself.

converted_dir
Path to the root directory of the converted dataset.

converted_exists() → bool
Return True if the saved dataset exists, otherwise False.

exists() → bool
Return True if the dataset exists, otherwise False.

load(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Load a file into a Music object.

on_the_fly() → FolderDatasetType
Enable on-the-fly mode and convert the data on the fly.

Returns

Return type Object itself.

read(*filename: Any*) → muspy.music.Music
Read a file into a Music object.

use_converted() → FolderDatasetType
Disable on-the-fly mode and use converted data.

Returns

Return type Object itself.

class muspy.datasets.**HymnalDataset**(*root: Union[str, pathlib.Path], download: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Hymnal Dataset.

download() → muspy.datasets.base.FolderDataset
Download the source datasets.

Returns

Return type Object itself.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**HymnalTuneDataset**(*root: Union[str, pathlib.Path], download: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Hymnal Dataset (tune only).

download() → muspy.datasets.base.FolderDataset
Download the source datasets.

Returns

Return type Object itself.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**JSBChoralesDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Johann Sebastian Bach Chorales Dataset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**LakhMIDIAlignedDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Lakh MIDI Dataset - aligned subset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**LakhMIDIDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Lakh MIDI Dataset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**LakhMIDIMatchedDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Lakh MIDI Dataset - matched subset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**MAESTRODatasetV1**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

MAESTRO Dataset (MIDI only).

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**MAESTRODatasetV2**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

MAESTRO Dataset (MIDI only).

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**Music21Dataset**(*composer: Optional[str] = None*)
A class of datasets containing files in music21 corpus.

Parameters

- **composer** (*str*) – Name of a composer or a collection. Please refer to the music21 corpus reference page for a full list [1].
- **extensions** (*list of str*) – File extensions of desired files.

References

[1] https://web.mit.edu/music21/doc/about/referenceCorpus.html

convert(*root: Union[str, pathlib.Path], kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True*) → muspy.datasets.base.MusicDataset
Convert and save the Music objects.

Parameters

- **root** (*str or Path*) – Root directory to save the data.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.
- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.

class muspy.datasets.**MusicDataset**(*root: Union[str, pathlib.Path], kind: str = 'json'*)

Class for datasets of MusPy JSON/YAML files.

root

Root directory of the dataset.

Type str or Path

kind

File format of the data. Defaults to 'json'.

Type {'json', 'yaml'}, optional

See also:

muspy.Dataset Base class for MusPy datasets.

class muspy.datasets.**NESMusicDatabase**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

NES Music Database.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music

Read a file into a Music object.

class muspy.datasets.**NottinghamDatabase**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Nottingham Database.

class muspy.datasets.**RemoteABCFolderDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)

Base class for remote datasets storing ABC files in a folder.

See also:

muspy.ABCFolderDataset Class for datasets storing ABC files in a folder.

muspy.RemoteDataset Base class for remote MusPy datasets.

class muspy.datasets.**RemoteDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False*)

Base class for remote MusPy datasets.

This class extends *muspy.Dataset* to support remote datasets. To build a custom remote dataset, please refer to the documentation of *muspy.Dataset* for details. In addition, set the class attribute `_sources` to the URLs to the source files (see Notes).

root

Root directory of the dataset.

Type str or Path

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.

Raises RuntimeError: – If `download_and_extract` is False but file `{root}/.muspy.success` does not exist (see below).

Important: `muspy.Dataset.exists()` depends solely on a special file named `.muspy.success` in directory `{root}/_converted/`. This file serves as an indicator for the existence and integrity of the dataset. It will automatically be created if the dataset is successfully downloaded and extracted by `muspy.Dataset.download_and_extract()`. If the dataset is downloaded manually, make sure to create the `.muspy.success` file in directory `{root}/_converted/` to prevent errors.

Notes

The class attribute `_sources` is a dictionary storing the following information of each source file.

- filename (str): Name to save the file.
- url (str): URL to the file.
- archive (bool): Whether the file is an archive.
- md5 (str, optional): Expected MD5 checksum of the file.

Here is an example.:

```
_sources = {
    "example": {
        "filename": "example.tar.gz",
        "url": "https://www.example.com/example.tar.gz",
        "archive": True,
        "md5": None,
    }
}
```

See also:

muspy.Dataset Base class for MusPy datasets.

download() → RemoteDatasetType
Download the source datasets.

Returns

Return type Object itself.

download_and_extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archives.

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

Notes

Equivalent to `RemoteDataset.download().extract(cleanup)`.

exists() → bool
Return True if the dataset exists, otherwise False.

extract(*cleanup: bool = False*) → RemoteDatasetType
Extract the downloaded archive(s).

Parameters **cleanup** (*bool, optional*) – Whether to remove the original archive. Defaults to False.

Returns

Return type Object itself.

source_exists() → bool
Return True if all the sources exist, otherwise False.

class muspy.datasets.**RemoteFolderDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Base class for remote datasets stoting files in a folder.

root
Root directory of the dataset.

Type str or Path

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.
- **convert** (*bool, optional*) – Whether to convert the dataset to MusPy JSON/YAML files. If False, will check if converted data exists. If so, disable on-the-fly mode. If not, enable on-the-fly mode and warns. Defaults to False.
- **kind** (*{'json', 'yaml'}, optional*) – File format to save the data. Defaults to 'json'.

- **n_jobs** (*int, optional*) – Maximum number of concurrently running jobs. If equal to 1, disable multiprocessing. Defaults to 1.
- **ignore_exceptions** (*bool, optional*) – Whether to ignore errors and skip failed conversions. This can be helpful if some source files are known to be corrupted. Defaults to True.
- **use_converted** (*bool, optional*) – Force to disable on-the-fly mode and use stored converted data

See also:

muspy.FolderDataset Class for datasets storing files in a folder.

muspy.RemoteDataset Base class for remote MusPy datasets.

read(*filename: str*) → muspy.music.Music
Read a file into a Music object.

class muspy.datasets.**RemoteMusicDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, kind: str = 'json'*)
Base class for remote datasets of MusPy JSON/YAML files.

root
Root directory of the dataset.

Type str or Path

kind
File format of the data. Defaults to 'json'.

Type {'json', 'yaml'}, optional

Parameters

- **download_and_extract** (*bool, optional*) – Whether to download and extract the dataset. Defaults to False.
- **cleanup** (*bool, optional*) – Whether to remove the original archive(s). Defaults to False.

See also:

muspy.MusicDataset Class for datasets of MusPy JSON/YAML files.

muspy.RemoteDataset Base class for remote MusPy datasets.

class muspy.datasets.**WikifoniaDataset**(*root: Union[str, pathlib.Path], download_and_extract: bool = False, cleanup: bool = False, convert: bool = False, kind: str = 'json', n_jobs: int = 1, ignore_exceptions: bool = True, use_converted: Optional[bool] = None*)
Wikifonia dataset.

read(*filename: Union[str, pathlib.Path]*) → muspy.music.Music
Read a file into a Music object.

muspy.datasets.**get_dataset**(*key: str*) → Type[muspy.datasets.base.Dataset]
Return a certain dataset class by key.

Parameters **key** (*str*) – Dataset key (case-insensitive).

Returns

Return type The corresponding dataset class.

muspy.datasets.**list_datasets**()
Return all supported dataset classes as a list.

Returns

Return type A list of all supported dataset classes.

7.10.3 muspy.external

External dependencies.

This module provides functions for working with external dependencies.

Functions

- download_bravura_font
- download_musescore_soundfont
- get_bravura_font_dir
- get_bravura_font_path
- get_musescore_soundfont_dir
- get_musescore_soundfont_path

muspy.external.**download_bravura_font**()
Download the Bravura font.

muspy.external.**download_musescore_soundfont**()
Download the MuseScore General soundfont.

muspy.external.**get_bravura_font_dir**() → pathlib.Path
Return path to the directory of the Bravura font.

muspy.external.**get_bravura_font_path**() → pathlib.Path
Return path to the Bravura font.

muspy.external.**get_musescore_soundfont_dir**() → pathlib.Path
Return path to the MuseScore General soundfont directory.

muspy.external.**get_musescore_soundfont_path**() → pathlib.Path
Return path to the MuseScore General soundfont.

7.10.4 muspy.inputs

Input interfaces.

This module provides input interfaces for common symbolic music formats, MusPy's native JSON and YAML formats, other symbolic music libraries and commonly-used representations in music generation.

Functions

- from_event_representation
- from_mido
- from_music21
- from_music21_opus
- from_note_representation
- from_object
- from_pianoroll_representation
- from_pitch_representation
- from_pretty_midi
- from_pypianoroll
- from_representation
- load
- load_json
- load_yaml
- read
- read_abc
- read_abc_string
- read_midi
- read_musicxml

Errors

- MIDIError
- MusicXMLError

exception muspy.inputs.**MIDIError**
An error class for MIDI related exceptions.

exception muspy.inputs.**MusicXMLError**
An error class for MusicXML related exceptions.

muspy.inputs.**from_event_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *max_time_shift: int = 100*, *velocity_bins: int = 32*, *default_velocity: int = 64*) → muspy.music.Music
Decode event-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in event-based representation to decode. Will be casted to integer if not of integer type.

- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_single_note_off_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, a note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use_end_of_sequence_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **max_time_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity_bins** (*int*) – Number of velocity bins to use. Defaults to 32.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.inputs.**from_mido**(*midi: mido.midifiles.midifiles.MidiFile*, *duplicate_note_mode: str = 'fifo'*) → muspy.music.Music

Return a mido MidiFile object as a Music object.

Parameters

- **midi** (mido.MidiFile) – Mido MidiFile object to convert.
- **duplicate_note_mode** (*{'fifo', 'lifo, 'close_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close_all': close all note on messages

Returns Converted Music object.

Return type *muspy.Music*

muspy.inputs.**from_music21**(*stream: music21.stream.Stream*, *resolution=24*) → Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]

Return a music21 Stream object as Music or Track object(s).

Parameters

- **stream** (*music21.stream.Stream*) – Stream object to convert.

- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music or Track object(s).

Return type *muspy.Music* or *muspy.Track*

muspy.inputs.**from_music21_opus**(*opus: music21.stream.Opus*, *resolution=24*) → List[muspy.music.Music]

Return a music21 Opus object as a list of Music objects.

Parameters

- **opus** (*music21.stream.Opus*) – Opus object to convert.
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object.

Return type *muspy.Music*

muspy.inputs.**from_note_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_start_end: bool = False*, *encode_velocity: bool = True*, *default_velocity: int = 64*) → muspy.music.Music

Decode note-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in note-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_start_end** (*bool*) – Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **default_velocity** (*int*) – Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.inputs.**from_object**(*obj: Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty_midi.pretty_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack], **kwargs*) → Union[muspy.music.Music, List[muspy.music.Music], muspy.classes.Track, List[muspy.classes.Track]]

Return an outside object as a Music object.

Parameters **obj** – Object to convert. Supported objects are *music21.Stream*, `mido.MidiTrack`, `pretty_midi.PrettyMIDI`, and `pypianoroll.Multitrack` objects.

Returns Converted Music object.

Return type *muspy.Music*

muspy.inputs.**from_pianoroll_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *encode_velocity: bool = True*, *default_velocity: int = 64*) → muspy.music.Music

Decode pitch-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in piano-roll representation to decode. Will be casted to integer if not of integer type. If *encode_velocity* is True, will be casted to boolean if not of boolean type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode velocities. Defaults to True.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.inputs.**from_pitch_representation**(*array: numpy.ndarray*, *resolution: int = 24*, *program: int = 0*, *is_drum: bool = False*, *use_hold_state: bool = False*, *default_velocity: int = 64*) → muspy.music.Music

Decode pitch-based representation into a Music object.

Parameters

- **array** (*ndarray*) – Array in pitch-based representation to decode. Will be casted to integer if not of integer type.
- **resolution** (*int*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.
- **program** (*int, optional*) – Program number according to General MIDI specification [1]. Acceptable values are 0 to 127. Defaults to 0 (Acoustic Grand Piano).
- **is_drum** (*bool, optional*) – A boolean indicating if it is a percussion track. Defaults to False.
- **use_hold_state** (*bool*) – Whether to use a special state for holds. Defaults to False.

- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns Decoded Music object.

Return type *muspy.Music*

References

[1] https://www.midi.org/specifications/item/gm-level-1-sound-set

muspy.inputs.**from_pretty_midi**(*midi: pretty_midi.pretty_midi.PrettyMIDI*) → muspy.music.Music

Return a pretty_midi PrettyMIDI object as a Music object.

Parameters **midi** (pretty_midi.PrettyMIDI) – PrettyMIDI object to convert.

Returns Converted Music object.

Return type *muspy.Music*

muspy.inputs.**from_pypianoroll**(*multitrack: pypianoroll.multitrack.Multitrack*, *default_velocity: int = 64*) → muspy.music.Music

Return a Pypianoroll Multitrack object as a Music object.

Parameters

- **multitrack** (pypianoroll.Multitrack) – Pypianoroll Multitrack object to convert.
- **default_velocity** (*int*) – Default velocity value to use when decoding. Defaults to 64.

Returns **music** – Converted MusPy Music object.

Return type *muspy.Music*

muspy.inputs.**from_representation**(*array: numpy.ndarray*, *kind: str*, ***kwargs*) → muspy.music.Music

Update with the given representation.

Parameters

- **array** (numpy.ndarray) – Array in a supported representation.
- **kind** (*str, {'pitch', 'pianoroll', 'event', 'note'}*) – Data representation type (case-insensitive).

Returns Converted Music object.

Return type *muspy.Music*

muspy.inputs.**load**(*path: Union[str, pathlib.Path]*, *kind: Optional[str] = None*, ***kwargs*) → muspy.music.Music

Load a JSON or a YAML file into a Music object.

Parameters

- **path** (*str or Path*) – Path to the file to load.
- **kind** (*{'json', 'yaml'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.
- ****kwargs** (*dict*) – Keyword arguments to pass to the target function. See *muspy.load_json()* or *muspy.load_yaml()* for available arguments.

Returns Loaded Music object.

Return type *muspy.Music*

See also:

muspy.read() Read a MIDI/MusicXML/ABC file into a Music object.

muspy.inputs.**load_json**(*path: Union[str, pathlib.Path]*) → muspy.music.Music
Load a JSON file into a Music object.

Parameters **path** (*str or Path*) – Path to the file to load.

Returns Loaded Music object.

Return type *muspy.Music*

muspy.inputs.**load_yaml**(*path: Union[str, pathlib.Path]*) → muspy.music.Music
Load a YAML file into a Music object.

Parameters **path** (*str or Path*) – Path to the file to load.

Returns Loaded Music object.

Return type *muspy.Music*

muspy.inputs.**read**(*path: Union[str, pathlib.Path], kind: Optional[str] = None, **kwargs*) → Union[muspy.music.Music, List[muspy.music.Music]]
Read a MIDI/MusicXML/ABC file into a Music object.

Parameters

- **path** (*str or Path*) – Path to the file to read.
- **kind** (*{'midi', 'musicxml', 'abc'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.

Returns Converted Music object(s).

Return type *muspy.Music* or list of *muspy.Music*

See also:

muspy.load() Load a JSON or a YAML file into a Music object.

muspy.inputs.**read_abc**(*path: Union[str, pathlib.Path], number: Optional[int] = None, resolution=24*) → List[muspy.music.Music]
Return an ABC file into Music object(s) using music21 backend.

Parameters

- **path** (*str or Path*) – Path to the ABC file to read.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object(s).

Return type list of *muspy.Music*

muspy.inputs.**read_abc_string**(*data_str: str, number: Optional[int] = None, resolution=24*)
Read ABC data into Music object(s) using music21 backend.

Parameters

- **data_str** (*str*) – ABC data to parse.
- **number** (*int*) – Reference number of a specific tune to read (i.e., the 'X:' field).
- **resolution** (*int, optional*) – Time steps per quarter note. Defaults to *muspy.DEFAULT_RESOLUTION*.

Returns Converted Music object(s).

Return type *muspy.Music*

muspy.inputs.**read_midi**(*path: Union[str, pathlib.Path], backend: str = 'mido', duplicate_note_mode: str = 'fifo'*) → muspy.music.Music

Read a MIDI file into a Music object.

Parameters

- **path** (*str or Path*) – Path to the MIDI file to read.
- **backend** (*{'mido', 'pretty_midi'}*) – Backend to use.
- **duplicate_note_mode** (*{'fifo', 'lifo, 'close_all'}*) – Policy for dealing with duplicate notes. When a note off message is presetned while there are multiple correspoding note on messages that have not yet been closed, we need a policy to decide which note on messages to close. Defaults to 'fifo'. Only used when *backend='mido'*.
  - 'fifo' (first in first out): close the earliest note on
  - 'lifo' (first in first out):close the latest note on
  - 'close_all': close all note on messages

Returns Converted Music object.

Return type *muspy.Music*

muspy.inputs.**read_musicxml**(*path: Union[str, pathlib.Path], compressed: Optional[bool] = None*) → muspy.music.Music

Read a MusicXML file into a Music object.

Parameters **path** (*str or Path*) – Path to the MusicXML file to read.

Returns Converted Music object.

Return type *muspy.Music*

Notes

Grace notes and unpitched notes are not supported.

7.10.5 muspy.metrics

Objective metrics.

This module provides common objective metrics in music generation. These objective metrics could be used to evaluate a music generation system by comparing the statistical difference between the training data and the generated samples.

Functions

- drum_in_pattern_rate
- drum_pattern_consistency
- empty_beat_rate
- empty_measure_rate
- groove_consistency
- n_pitch_classes_used
- n_pitches_used
- pitch_class_entropy
- pitch_entropy
- pitch_in_scale_rate
- pitch_range
- polyphony
- polyphony_rate
- scale_consistency

muspy.metrics.**drum_in_pattern_rate**(*music: muspy.music.Music*, *meter: str*) → float
Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$drum\_in\_pattern\_rate = \frac{\#(drum\_notes\_in\_pattern)}{\#(drum\_notes)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **meter** (*str, {'duple', 'triple'}*) – Meter of the drum pattern.

Returns Drum-in-pattern rate.

Return type float

See also:

muspy.drum_pattern_consistency() Compute the largest drum-in-pattern rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.metrics.**drum_pattern_consistency**(*music: muspy.music.Music*) → float
Return the largest drum-in-pattern rate.

The drum pattern consistency is defined as the largest drum-in-pattern rate over duple and triple meters. Only drum tracks are considered. Return NaN if no drum note is found.

$$drum\_pattern\_consistency = \max_{meter} drum\_in\_pattern\_rate(meter)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Drum pattern consistency.

Return type float

See also:

muspy.drum_in_pattern_rate() Compute the ratio of drum notes in a certain drum pattern.

muspy.metrics.**empty_beat_rate**(*music: muspy.music.Music*) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero. This metric is also implemented in Pypianoroll [1].

$$empty\_beat\_rate = \frac{\#(empty\_beats)}{\#(beats)}$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Empty-beat rate.

Return type float

See also:

muspy.empty_measure_rate() Compute the ratio of empty measures.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, "Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls," in Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR), 2018.

muspy.metrics.**empty_measure_rate**(*music: muspy.music.Music*, *measure_resolution: int*) → float

Return the ratio of empty measures.

The empty-measure rate is defined as the ratio of the number of empty measures (where no note is played) to the total number of measures. Note that this metric only works for songs with a constant time signature. Return NaN if song length is zero. This metric is used in [1].

$$empty\_measure\_rate = \frac{\#(empty\_measures)}{\#(measures)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure_resolution** (*int*) – Time steps per measure.

Returns Empty-measure rate.

Return type float

See also:

muspy.empty_beat_rate() Compute the ratio of empty beats.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.metrics.**groove_consistency**(*music: muspy.music.Music*, *measure_resolution: int*) → float

Return the groove consistency.

The groove consistency is defined as the mean hamming distance of the neighboring measures.

$$groove\_consistency = 1 - \frac{1}{T-1}\sum_{i=1}^{T-1} d(G_i, G_{i+1})$$

Here, T is the number of measures, G_i is the binary onset vector of the i-th measure (a one at position that has an onset, otherwise a zero), and $d(G, G')$ is the hamming distance between two vectors G and G'. Note that this metric only works for songs with a constant time signature. Return NaN if the number of measures is less than two. This metric is used in [1].

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **measure_resolution** (*int*) – Time steps per measure.

Returns Groove consistency.

Return type float

References

1. Shih-Lun Wu and Yi-Hsuan Yang, "The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures", in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

muspy.metrics.**n_pitch_classes_used**(*music: muspy.music.Music*) → int

Return the number of unique pitch classes used.

Drum tracks are ignored.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Number of unique pitch classes used.

Return type int

See also:

muspy.n_pitches_used() Compute the number of unique pitches used.

muspy.metrics.**n_pitches_used**(*music: muspy.music.Music*) → int

Return the number of unique pitches used.

Drum tracks are ignored.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Number of unique pitch used.

Return type int

See also:

muspy.n_pitch_class_used() Compute the number of unique pitch classes used.

muspy.metrics.**pitch_class_entropy**(*music: muspy.music.Music*) → float
Return the entropy of the normalized note pitch class histogram.

The pitch class entropy is defined as the Shannon entropy of the normalized note pitch class histogram. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_class\_entropy = -\sum_{i=0}^{11} P(pitch\_class = i) \times \log_2 P(pitch\_class = i)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch class entropy.

Return type float

See also:

muspy.pitch_entropy() Compute the entropy of the normalized pitch histogram.

References

1. Shih-Lun Wu and Yi-Hsuan Yang, "The Jazz Transformer on the Front Line: Exploring the Shortcomings of AI-composed Music through Quantitative Measures", in Proceedings of the 21st International Society for Music Information Retrieval Conference, 2020.

muspy.metrics.**pitch_entropy**(*music: muspy.music.Music*) → float
Return the entropy of the normalized note pitch histogram.

The pitch entropy is defined as the Shannon entropy of the normalized note pitch histogram. Drum tracks are ignored. Return NaN if no note is found.

$$pitch\_entropy = -\sum_{i=0}^{127} P(pitch = i) \log_2 P(pitch = i)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch entropy.

Return type float

See also:

muspy.pitch_class_entropy() Compute the entropy of the normalized pitch class histogram.

muspy.metrics.**pitch_in_scale_rate**(*music: muspy.music.Music*, *root: int*, *mode: str*) → float
Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$pitch\_in\_scale\_rate = \frac{\#(notes\_in\_scale)}{\#(notes)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str, {'major', 'minor'}*) – Mode of the scale.

Returns Pitch-in-scale rate.

Return type float

See also:

muspy.scale_consistency() Compute the largest pitch-in-class rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.metrics.**pitch_range**(*music: muspy.music.Music*) → int

Return the pitch range.

Drum tracks are ignored. Return zero if no note is found.

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Pitch range.

Return type int

muspy.metrics.**polyphony**(*music: muspy.music.Music*) → float

Return the average number of pitches being played concurrently.

The polyphony is defined as the average number of pitches being played at the same time, evaluated only at time steps where at least one pitch is on. Drum tracks are ignored. Return NaN if no note is found.

$$polyphony = \frac{\#(pitches\_when\_at\_least\_one\_pitch\_is\_on)}{\#(time\_steps\_where\_at\_least\_one\_pitch\_is\_on)}$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Polyphony.

Return type float

See also:

muspy.polyphony_rate() Compute the ratio of time steps where multiple pitches are on.

muspy.metrics.**polyphony_rate**(*music: muspy.music.Music*, *threshold: int = 2*) → float

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$polyphony\_rate = \frac{\#(time\_steps\_where\_multiple\_pitches\_are\_on)}{\#(time\_steps)}$$

Parameters

- **music** (*muspy.Music*) – Music object to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

Returns Polyphony rate.

Return type float

See also:

muspy.polyphony() Compute the average number of pitches being played at the same time.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

muspy.metrics.**scale_consistency**(*music: muspy.music.Music*) → float

Return the largest pitch-in-scale rate.

The scale consistency is defined as the largest pitch-in-scale rate over all major and minor scales. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$scale\_consistency = \max_{root,mode} pitch\_in\_scale\_rate(root, mode)$$

Parameters **music** (*muspy.Music*) – Music object to evaluate.

Returns Scale consistency.

Return type float

See also:

muspy.pitch_in_scale_rate() Compute the ratio of pitches in a certain musical scale.

References

1. Olof Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training," in NeuIPS Workshop on Constructive Machine Learning, 2016.

7.10.6 muspy.outputs

Output interfaces.

This module provides output interfaces for common symbolic music formats, MusPy's native JSON and YAML formats, other symbolic music libraries and commonly-used representations in music generation.

Functions

- save
- save_json
- save_yaml
- to_event_representation

- to_mido
- to_music21
- to_note_representation
- to_object
- to_pianoroll_representation
- to_pitch_representation
- to_pretty_midi
- to_pypianoroll
- to_representation
- write
- write_abc
- write_audio
- write_midi
- write_musicxml

muspy.outputs.**save**(*path: Union[str, pathlib.Path], music: Music, kind: Optional[str] = None, **kwargs*)

Save a Music object loselessly to a JSON or a YAML file.

Parameters

- **path** (*str or Path*) – Path to save the file.
- **music** (*muspy.Music*) – Music object to save.
- **kind** (*{'json', 'yaml'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.

See also:

muspy.write() Write a Music object to a MIDI/MusicXML/ABC/audio file.

Notes

The conversion can be lossy if any nonserializable object is used (for example, an Annotation object, which can store data of any type).

muspy.outputs.**save_json**(*path: Union[str, pathlib.Path], music: Music*)

Save a Music object to a JSON file.

Parameters

- **path** (*str or Path*) – Path to save the JSON file.
- **music** (*muspy.Music*) – Music object to save.

muspy.outputs.**save_yaml**(*path: Union[str, pathlib.Path], music: Music*)

Save a Music object to a YAML file.

Parameters

- **path** (*str or Path*) – Path to save the YAML file.
- **music** (*muspy.Music*) – Music object to save.

muspy.outputs.**to_event_representation**(*music: Music*, *use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *encode_velocity: bool = False*, *force_velocity_event: bool = True*, *max_time_shift: int = 100*, *velocity_bins: int = 32*) → numpy.ndarray

Encode a Music object into event-based representation.

The event-based represetantion represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is M x 1, where M is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-one events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_single_note_off_event** (*bool*) – Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.
- **use_end_of_sequence_event** (*bool*) – Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode velocities.
- **force_velocity_event** (*bool*) – Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.
- **max_time_shift** (*int*) – Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.
- **velocity_bins** (*int*) – Number of velocity bins to use. Defaults to 32.

Returns Encoded array in event-based representation.

Return type ndarray, dtype=uint16, shape=(?, 1)

muspy.outputs.**to_mido**(*music: Music*, *use_note_on_as_note_off: bool = True*)

Return a Music object as a MidiFile object.

Parameters

- **music** (*muspy.Music* object) – Music object to convert.
- **use_note_on_as_note_off** (*bool*) – Whether to use a note on message with zero velocity instead of a note off message.

Returns Converted MidiFile object.

Return type mido.MidiFile

muspy.outputs.**to_music21**(*music: Music*) → music21.stream.Score

Return a Music object as a music21 Score object.

Parameters **music** (*muspy.Music*) – Music object to convert.

Returns Converted music21 Score object.

Return type *music21.stream.Score*

muspy.outputs.**to_note_representation**(*music: Music*, *use_start_end: bool = False*, *encode_velocity: bool = True*, *dtype: Union[numpy.dtype, type, str] = <class 'int'>*) → numpy.ndarray

Encode a Music object into note-based representation.

The note-based represetantion represents music as a sequence of (time, pitch, duration, velocity) tuples. For example, a note Note(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 60, 4, 64). The output shape is N * D, where N is the number of notes and D is 4 when *encode_velocity* is True, otherwise D is 3. The values of the second dimension represent time, pitch, duration and velocity (discarded when *encode_velocity* is False).

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_start_end** (*bool*) – Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.
- **encode_velocity** (*bool*) – Whether to encode note velocities. Defaults to True.
- **dtype** (*dtype, type or str*) – Data type of the return array. Defaults to int.

Returns Encoded array in note-based representation.

Return type ndarray, shape=(?, 3 or 4)

muspy.outputs.**to_object**(*music: Music*, *kind: str*, ***kwargs*) → Union[music21.stream.Stream, mido.midifiles.midifiles.MidiFile, pretty_midi.pretty_midi.PrettyMIDI, pypianoroll.multitrack.Multitrack]

Return a Music object as an object in other libraries.

Supported classes are *music21.Stream*, mido.MidiTrack, pretty_midi.PrettyMIDI and pypianoroll.Multitrack.

Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str, {'music21', 'mido', 'pretty_midi', 'pypianoroll'}*) – Target class (case-insensitive).

Returns Converted object.

Return type *music21.Stream*, mido.MidiTrack, pretty_midi.PrettyMIDI or pypianoroll.Multitrack

muspy.outputs.**to_pianoroll_representation**(*music: Music*, *encode_velocity: bool = True*) → numpy.ndarray

Encode notes into piano-roll representation.

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **encode_velocity** (*bool*) – Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

Returns Encoded array in piano-roll representation.

Return type ndarray, dtype=uint8 or bool, shape=(?, 128)

muspy.outputs.**to_pitch_representation**(*music: Music*, *use_hold_state: bool = False*) → numpy.ndarray

Encode a Music object into pitch-based representation.

The pitch-based represetantion represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

Parameters

- **music** (*muspy.Music*) – Music object to encode.
- **use_hold_state** (*bool*) – Whether to use a special state for holds. Defaults to False.

Returns Encoded array in pitch-based representation.

Return type ndarray, dtype=uint8, shape=(?, 1)

muspy.outputs.**to_pretty_midi**(*music: Music*) → pretty_midi.pretty_midi.PrettyMIDI
Return a Music object as a PrettyMIDI object.

Tempo changes are not supported yet.

Parameters **music** (*muspy.Music* object) – Music object to convert.

Returns Converted PrettyMIDI object.

Return type pretty_midi.PrettyMIDI

muspy.outputs.**to_pypianoroll**(*music: Music*) → pypianoroll.multitrack.Multitrack
Return a Music object as a Multitrack object.

Parameters **music** (*muspy.Music*) – Music object to convert.

Returns **multitrack** – Converted Multitrack object.

Return type pypianoroll.Multitrack

muspy.outputs.**to_representation**(*music: Music*, *kind: str*, ***kwargs*) → numpy.ndarray
Return a Music object in a specific representation.

Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str, {'pitch', 'piano-roll', 'event', 'note'}*) – Target representation (case-insensitive).

Returns **array** – Converted representation.

Return type ndarray

muspy.outputs.**synthesize**(*music: Music*, *soundfont_path: Union[str, pathlib.Path, None] = None*, *rate: int = 44100*) → numpy.ndarray
Synthesize a Music object to raw audio.

Parameters

- **music** (*muspy.Music*) – Music object to write.
- **soundfont_path** (*str or Path, optional*) – Path to the soundfount file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.

Returns Synthesized waveform.

Return type ndarray, dtype=int16, shape=(?, 2)

muspy.outputs.**write**(*path: Union[str, pathlib.Path], music: Music, kind: Optional[str] = None, **kwargs*)

Write a Music object to a MIDI/MusicXML/ABC/audio file.

Parameters

- **path** (*str or Path*) – Path to write the file.
- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*{'midi', 'musicxml', 'abc', 'audio'}, optional*) – Format to save (case-insensitive). Defaults to infer the format from the extension.

See also:

muspy.save() Save a Music object loselessly to a JSON or a YAML file.

muspy.outputs.**write_abc**(*path: Union[str, pathlib.Path], music: Music*)

Write a Music object to a ABC file.

Parameters

- **path** (*str or Path*) – Path to write the ABC file.
- **music** (*muspy.Music*) – Music object to write.

muspy.outputs.**write_audio**(*path: Union[str, pathlib.Path], music: Music, soundfont_path: Union[str, pathlib.Path, None] = None, rate: int = 44100, audio_format: Optional[str] = None*)

Write a Music object to an audio file.

Supported formats include WAV, AIFF, FLAC and OGA.

Parameters

- **path** (*str or Path*) – Path to write the audio file.
- **music** (*muspy.Music*) – Music object to write.
- **soundfont_path** (*str or Path, optional*) – Path to the soundfount file. Defaults to the path to the downloaded MuseScore General soundfont.
- **rate** (*int*) – Sample rate (in samples per sec). Defaults to 44100.
- **audio_format** (*str, {'wav', 'aiff', 'flac', 'oga'}, optional*) – File format to write. If None, infer it from the extension.

muspy.outputs.**write_midi**(*path: Union[str, pathlib.Path], music: Music, backend: str = 'mido', **kwargs*)

Write a Music object to a MIDI file.

Parameters

- **path** (*str or Path*) – Path to write the MIDI file.
- **music** (*muspy.Music*) – Music object to write.
- **backend** (*{'mido', 'pretty_midi'}*) – Backend to use. Defaults to 'mido'.

muspy.outputs.**write_musicxml**(*path: Union[str, pathlib.Path], music: Music, compressed: Optional[bool] = None*)

Write a Music object to a MusicXML file.

Parameters

- **path** (*str or Path*) – Path to write the MusicXML file.

- **music** (*muspy.Music*) – Music object to write.
- **compressed** (*bool, optional*) – Whether to write to a compressed MusicXML file. If None, infer from the extension of the filename ('.xml' and '.musicxml' for an uncompressed file, '.mxl' for a compressed file).

7.10.7 muspy.processors

Representation processors.

This module defines the processors for commonly used representations.

Classes

- NoteRepresentationProcessor
- EventRepresentationProcessor
- PianoRollRepresentationProcessor
- PitchRepresentationProcessor

class muspy.processors.**NoteRepresentationProcessor**(*use_start_end: bool = False*, *encode_velocity: bool = True*, *dtype: Union[numpy.dtype, type, str] = <class 'int'>*, *default_velocity: int = 64*)

Note-based representation processor.

The note-based represetantion represents music as a sequence of (pitch, time, duration, velocity) tuples. For example, a note Note(time=0, duration=4, pitch=60, velocity=64) will be encoded as a tuple (0, 4, 60, 64). The output shape is L * D, where L is th number of notes and D is 4 when *encode_velocity* is True, otherwise D is 3. The values of the second dimension represent pitch, time, duration and velocity (discarded when *encode_velocity* is False).

use_start_end
Whether to use 'start' and 'end' to encode the timing rather than 'time' and 'duration'. Defaults to False.

Type bool

encode_velocity
Whether to encode note velocities. Defaults to True.

Type bool

dtype
Data type of the return array. Defaults to int.

Type dtype, type or str

default_velocity
Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode note-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in note-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_note_representation() Return a Music object converted from note-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into note-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in note-based representation.

Return type ndarray (np.uint8)

See also:

muspy.to_note_representation() Convert a Music object into note-based representation.

class muspy.processors.**EventRepresentationProcessor**(*use_single_note_off_event: bool = False*, *use_end_of_sequence_event: bool = False*, *encode_velocity: bool = False*, *force_velocity_event: bool = True*, *max_time_shift: int = 100*, *velocity_bins: int = 32*, *default_velocity: int = 64*)
Event-based representation processor.

The event-based represetantion represents music as a sequence of events, including note-on, note-off, time-shift and velocity events. The output shape is M x 1, where M is the number of events. The values encode the events. The default configuration uses 0-127 to encode note-one events, 128-255 for note-off events, 256-355 for time-shift events, and 356 to 387 for velocity events.

use_single_note_off_event
Whether to use a single note-off event for all the pitches. If True, the note-off event will close all active notes, which can lead to lossy conversion for polyphonic music. Defaults to False.

Type bool

use_end_of_sequence_event
Whether to append an end-of-sequence event to the encoded sequence. Defaults to False.

Type bool

encode_velocity
Whether to encode velocities.

Type bool

force_velocity_event
Whether to add a velocity event before every note-on event. If False, velocity events are only used when the note velocity is changed (i.e., different from the previous one). Defaults to True.

Type bool

max_time_shift
Maximum time shift (in ticks) to be encoded as an separate event. Time shifts larger than *max_time_shift* will be decomposed into two or more time-shift events. Defaults to 100.

Type int

velocity_bins
Number of velocity bins to use. Defaults to 32.

Type int

default_velocity
Default velocity value to use when decoding. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode event-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in event-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_event_representation() Return a Music object converted from event-based representation.

encode(*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into event-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in event-based representation.

Return type ndarray (np.uint16)

See also:

muspy.to_event_representation() Convert a Music object into event-based representation.

class muspy.processors.**PianoRollRepresentationProcessor**(*encode_velocity: bool = True*, *default_velocity: int = 64*)
Piano-roll representation processor.

The piano-roll represetantion represents music as a time-pitch matrix, where the columns are the time steps and the rows are the pitches. The values indicate the presence of pitches at different time steps. The output shape is T x 128, where T is the number of time steps.

encode_velocity
Whether to encode velocities. If True, a binary-valued array will be return. Otherwise, an integer array will be return. Defaults to True.

Type bool

default_velocity
Default velocity value to use when decoding if *encode_velocity* is False. Defaults to 64.

Type int

decode(*array: numpy.ndarray*) → muspy.music.Music
Decode piano-roll representation into a Music object.

Parameters **array** (*ndarray*) – Array in piano-roll representation to decode. Cast to integer if not of integer type. If *encode_velocity* is True, casted to boolean if not of boolean type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_pianoroll_representation() Return a Music object converted from piano-roll representation.

encode (*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into piano-roll representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in piano-roll representation.

Return type ndarray (np.uint8)

See also:

muspy.to_pianoroll_representation() Convert a Music object into piano-roll representation.

class muspy.processors.**PitchRepresentationProcessor**(*use_hold_state: bool = False*, *default_velocity: int = 64*)
Pitch-based representation processor.

The pitch-based represetantion represents music as a sequence of pitch, rest and (optional) hold tokens. Only monophonic melodies are compatible with this representation. The output shape is T x 1, where T is the number of time steps. The values indicate whether the current time step is a pitch (0-127), a rest (128) or (optionally) a hold (129).

use_hold_state
Whether to use a special state for holds. Defaults to False.

Type bool

default_velocity
Default velocity value to use when decoding. Defaults to 64.

Type int

decode (*array: numpy.ndarray*) → muspy.music.Music
Decode pitch-based representation into a Music object.

Parameters **array** (*ndarray*) – Array in pitch-based representation to decode. Cast to integer if not of integer type.

Returns Decoded Music object.

Return type *muspy.Music* object

See also:

muspy.from_pitch_representation() Return a Music object converted from pitch-based representation.

encode (*music: muspy.music.Music*) → numpy.ndarray
Encode a Music object into pitch-based representation.

Parameters **music** (*muspy.Music* object) – Music object to encode.

Returns Encoded array in pitch-based representation.

Return type ndarray (np.uint8)

See also:

muspy.to_pitch_representation() Convert a Music object into pitch-based representation.

7.10.8 muspy.schemas

JSON, YAML and MusicXML schemas.

This module provide functions for working with MusPy's JSON and YAML schemas and the MusicXML schema.

Functions

- get_json_schema_path
- get_musicxml_schema_path
- get_yaml_schema_path

Variables

- DEFAULT_SCHEMA_VERSION

muspy.schemas.**get_json_schema_path**() → str
Return the path to the JSON schema.

muspy.schemas.**get_musicxml_schema_path**() → str
Return the path to the MusicXML schema.

muspy.schemas.**get_yaml_schema_path**() → str
Return the path to the YAML schema.

muspy.schemas.**validate_json**(*path: Union[str, pathlib.Path]*)
Validate a file against the JSON schema.

Parameters **path** (*str or Path*) – Path to the file to validate.

muspy.schemas.**validate_musicxml**(*path: Union[str, pathlib.Path]*)
Validate a file against the MusicXML schema.

Parameters **path** (*str or Path*) – Path to the file to validate.

muspy.schemas.**validate_yaml**(*path: Union[str, pathlib.Path]*)
Validate a file against the YAML schema.

Parameters **path** (*str or Path*) – Path to the file to validate.

7.10.9 muspy.visualization

Visualization tools.

This module provides functions for visualizing a Music object.

Classes

- ScorePlotter

Functions

- show
- show_pianoroll
- show_score

muspy.visualization.**show**(*music: Music*, *kind: str*, ***kwargs*)
Show visualization.

Parameters

- **music** (*muspy.Music*) – Music object to convert.
- **kind** (*str, {'piano-roll', 'score'}*) – Target representation.

muspy.visualization.**show_pianoroll**(*music: Music*, ***kwargs*)
Show pianoroll visualization.

muspy.visualization.**show_score**(*music: Music*, *figsize: Optional[Tuple[float, float]] = None*, *clef: str = 'treble'*, *clef_octave: Optional[int] = 0*, *note_spacing: Optional[int] = None*, *font_path: Union[str, pathlib.Path, None] = None*, *font_scale: Optional[float] = None*) → muspy.visualization.score.ScorePlotter
Show score visualization.

Parameters

- **music** (*muspy.Music*) – Music object to show.
- **figsize** (*(float, float), optional*) – Width and height in inches. Defaults to Matplotlib configuration.
- **clef** (*str, {'treble', 'alto', 'bass'}*) – Clef type. Defaults to a treble clef.
- **clef_octave** (*int*) – Clef octave. Defaults to zero.
- **note_spacing** (*int, optional*) – Spacing of notes. Defaults to 4.
- **font_path** (*str or Path, optional*) – Path to the music font. Defaults to the path to the built-in Bravura font.
- **font_scale** (*float, optional*) – Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

Returns A ScorePlotter object that handles the score.

Return type *muspy.ScorePlotter*

class muspy.visualization.**ScorePlotter**(*fig: matplotlib.figure.Figure*, *ax: matplotlib.axes._axes.Axes*, *resolution: int*, *note_spacing: Optional[int] = None*, *font_path: Union[str, pathlib.Path, None] = None*, *font_scale: Optional[float] = None*)
A plotter that handles the score visualization.

fig
Figure object to plot the score on.

Type matplotlib.figure.Figure

axes
Axes object to plot the score on.

Type matplotlib.axes.Axes

resolution
Time steps per quarter note.

Type int

note_spacing
Spacing of notes. Defaults to 4.

Type int, optional

font_path
Path to the music font. Defaults to the path to the downloaded Bravura font.

Type str or Path, optional

font_scale
Font scaling factor for finetuning. Defaults to 140, optimized for the Bravura font.

Type float, optional

adjust_fonts(*scale: Optional[float] = None*)
Adjust the fonts.

plot_bar_line() → matplotlib.lines.Line2D
Plot a bar line.

plot_clef(*kind='treble', octave=0*) → matplotlib.text.Text
Plot a clef.

plot_final_bar_line() → List[matplotlib.artist.Artist]
Plot an ending bar line.

plot_key_signature(*root: int, mode: str*)
Plot a key signature. Only major and minor keys are supported.

plot_note(*time, duration, pitch*) → Optional[Tuple[List[matplotlib.text.Text], List[matplotlib.patches.Arc]]]
Plot a note.

plot_object(*obj*)
Plot an object.

plot_staffs(*start: Optional[float] = None, end: Optional[float] = None*) → List[matplotlib.lines.Line2D]
Plot the staffs.

plot_tempo(*qpm*) → List[matplotlib.artist.Artist]
Plot a tempo as a metronome mark.

plot_time_signature(*numerator: int, denominator: int*) → List[matplotlib.text.Text]
Plot a time signature.

set_baseline(*y*)
Set baseline position (the y-coordinate of the first staff line).

update_boundaries(*left: Optional[float] = None, right: Optional[float] = None, bottom: Optional[float] = None, top: Optional[float] = None*)
Update boundaries.

Python Module Index

m

Index

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W